

AD-A196 290

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 88-89	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN AND PERFORMANCE ANALYSIS OF A RELATIONAL REPLICATED DATABASE SYSTEM		5. TYPE OF REPORT & PERIOD COVERED MS THESIS
6. AUTHOR(s) JON GREGORY HANSON		6. PERFORMING ORG. REPORT NUMBER
7. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: UNIVERSITY OF CENTRAL FLORIDA		8. CONTRACT OR GRANT NUMBER(s)
9. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) AFIT/NR Wright-Patterson AFB OH 45433-6583		12. REPORT DATE 1988
		13. NUMBER OF PAGES 418
		14. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) DISTRIBUTED UNLIMITED: APPROVED FOR PUBLIC RELEASE		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) SAME AS REPORT		
18. SUPPLEMENTARY NOTES Approved for Public Release: IAW AFR 190-1 LYNN E. WOLAYER <i>Lynn Wolayer</i> 19 July 88 Dean for Research and Professional Development Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED		

DESIGN AND PERFORMANCE ANALYSIS OF
A RELATIONAL REPLICATED DATABASE SYSTEM

by

JON GREGORY HANSON

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
the Department of Computer Science at
the University of Central Florida
Orlando, Florida

December 1987

Major Professor: Ali Orooji

Copyright 1987

by

Jon Gregory Hanson

ABSTRACT

The hardware organization and software structure of a new database system are presented. This system, the relational replicated database system (RRDS), is based on a set of replicated processors operating on a partitioned database. Performance improvements and capacity growth can be obtained by adding more processors to the configuration.

Based on design goals a set of hardware and software design questions were developed. The system then evolved according to a five-phase process, based on simulation and analysis, which addressed and resolved the design questions. Strategies and algorithms were developed for data access, data placement, and directory management for the hardware organization. A predictive performance analysis was conducted to determine the extent to which original design goals were satisfied. The predictive performance results, along with an analytical comparison with three other relational multi-backend systems, provided information about the strengths and weaknesses of our design as well as a basis for future research.

To my loving wife

Linda

ACKNOWLEDGEMENTS

I express my deep appreciation to Dr. Ali Orooji, my research advisor who provided valuable guidance and encouragement. It has been a privilege and pleasure to work under him.

I am also obliged to the other members of my committee: Dr. Larry Cottrell, Dr. Mostafa Bassiouni, Dr. H. N. Srinidhi, and Dr. Gary Whitehouse whose constructive comments helped me to improve the quality of this work. Special thanks are due to Dr. Cottrell for his guidance and encouragement for the last four years.

Finally, I wish to thank my wife, Linda, for her love and support, and my son Matthew, for adding joy and new dimension to my life.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	xii
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 SURVEY OF THE LITERATURE	4
A Classification of The Existing Database Systems	4
Conventional Database Systems	6
Distributed Database Systems	7
Backend Database Systems	11
Hardware-Based Backend Systems	11
Software-Based Backend Systems	13
Relational Multi-Backend Architectures	16
DIRECT	16
Relational Database Machine (RDBM)	21
Stonebraker's Machine	25
Teradata DBC/1012	28
DBMAC	31
CHAPTER 3 THE RRDS DESIGN PROCESS	35
Research Goals	35
Hardware Architectural Design Considerations	36
Selecting a Data Model and A Data Manipulation	
Language	37
The Design Process	52
CHAPTER 4 DEVELOPING A PRELIMINARY HARDWARE ARCHITECTURE	59
A Generic Software Multi-Backend Architecture	59
A Preliminary Architecture	65
Comparison of The RRDS Architecture With Alternative	
Approaches	68
Modeling Approach, Common Assumptions And	
Parameters	69
Modeling Approach	69
Common Assumptions	76
Parameters and Variables	79
An SIMD Architecture Model	80
An MIMD Architecture Model	87
A Functional Specialization Architecture Model	97
The RRDS Architecture Model	105
Workload Model and Experimentation Plan	111
A Comparison of The Four Architectures	113
CHAPTER 5 DATA ACCESS IN RRDS	123
A Clustered Data Access Strategy For RRDS	124
Clustering Concepts and Terminology	125

Data Access Based Upon A Clustering Scheme	126
Data Structures to Support Clustering	129
Algorithms For RRDS Clustering Scheme Directory Management	137
A Hierarchical Indexing Scheme Based Upon B+Trees	147
B+Tree Concepts and Terminology	148
Data Access Based Upon A B+Tree Hierarchical Index	151
Data Structures Based Upon A B+Tree Hierarchical Index	151
Algorithms For RRDS Hierarchical Index Directory Management	157
Selecting A Data Access Strategy	163
CHAPTER 6 DATA PLACEMENT IN RRDS	174
Different Approaches To Data Placement	174
Arbitrary Data Placement	175
Round Robin (RR) Data Placement	177
Value Range Partitioning (VRP) Data Placement	179
Analysis of RR and VRP	188
A Data Placement Strategy For RRDS	192
CHAPTER 7 DIRECTORY MANAGEMENT IN RRDS	195
Alternative Strategies Under Consideration	196
Controller Directory Management	196
Dedicated RC Directory Management	197
Rotating Directory Management	197
Rotating Directory Management Without Controller	198
Partitioned and Parallel Processed Directory Management	198
A Comparison Of RWOC and PPP Strategies	199
A Directory Management Strategy For RRDS	208
CHAPTER 8 RRDS PERFORMANCE ANALYSIS	209
Query Processing In RRDS	210
One-Relation Queries	210
The Select and Project Operations	211
The Insert Operation	212
The Delete Operation	214
The Update Operation	214
Aggregate Operations	215
Min and Max	215
Count, Sum, and Average	216
Two-Relation Queries	217
The Union Operation	217
Difference And Intersection Operations	218
The Join Operation	219
Performance Analysis	220
Performance Analysis Goals, Methodology, And Approach	220
The RRDS Simulation Model	223
The Queuing Network Model	223
The Hardware Parametric Model	231

The Workload Model	231
Experimentation Plan	231
Experimentation And Results	236
Conclusions	275
 CHAPTER 9 A PERFORMANCE EVALUATION OF RRDS WITH RESPECT TO THREE OTHER MULTI-BACKEND DATABASE SYSTEMS	278
The Architectures Used In The Comparison Study	279
The Parametric Model	282
Performance Comparisons	284
Selection Queries	285
Join Queries	290
Aggregate Function Queries	295
 CHAPTER 10 SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH	306
Summary	306
Conclusions	310
Suggestions For Future Research	313
 APPENDIX A RRDS DML SYNTAX	315
 APPENDIX B RRDS DFD DESIGN METHODOLOGY MODEL	319
 APPENDIX C SLAM NETWORKS FOR ARCHITECTURAL COMPARISON ANALYSIS	228
 APPENDIX D SLAM NETWORKS FOR DATA ACCESS ANALYSIS	366
 APPENDIX E SLAM NETWORKS FOR DATA PLACEMENT ANALYSIS	376
 APPENDIX F SLAM NETWORKS FOR DIRECTORY MANAGEMENT ANALYSIS	388
 APPENDIX G SLAM NETWORKS FOR PERFORMANCE ANALYSIS	399
 APPENDIX H VITA	411
 BIBLIOGRAPHY	413

LIST OF TABLES

1. RRDS Design Goals	37
2. Multi-Backend Database System Hardware Design Questions	38
3. The RRDS Operations	39
4. Architectural Comparison Workload Model	113
5. Architectural Comparison Experimentation Plan . .	114
6. Fixed Query - Response Time	115
7. Effect Of Query Interarrival Time On Response Time	118
8. Effect On Response Time Of Different Query Mixes	119
9. Increased Database Size - Response Time	120
10. Data Access Strategy Experimentation Plan	165
11. Data Access Strategy Workload Model	166
12. Average Response Time For Alternative Data Access Strategies	167
13. Effect Of Predicate Types	169
14. Response Time For Clustering Scheme For Variable Clusters/Relation	171
15. Effect On Response Time Of Increasing B+_Tree Degree	171
16. Response Time For Select Query As Degree Of B+_Tree Increases And Database Size Increases . .	172
17. Directory Size In Bytes For B+_Tree Data Access .	173
18. Effect Of Number Of Partitioning Attributes On Insert Response Time	189
19. Effect Of Descriptors/Partitioning Attribute On Insert Response Time	189

20. Average Response Time For Alternative Directory Management Strategies	202
21. Response Times For Directory Management Strategies for Variable Records/Relation	206
22. Effect Of Predicate Types On Different Directory Management Strategies	207
23. RRDS Simulation Model Hardware Parameters	232
24. The Workload Parameter Set	233
25. RRDS Performance Analysis Experimentation Plan .	235
26. Average Response Time For The Three RRDS Configurations	238
27. Percentage Ideal Goal For Experiment 1	239
28. Effect Of Relation Cardinality On Response Time .	241
29. Percentage Ideal Goal For Experiment 2	243
30. Effect Of Response Set Size On Response Time .	244
31. Percentage Ideal Goal For Experiment 3	247
32. Effect Of Number Of Predicates On Select Response Time	248
33. Percentage Ideal Goal For Experiment 4	249
34. Effect Of Predicate Type On Select Response Time	249
35. Percentage Ideal Goal For Experiment 5	250
36. Percentage Ideal Goal For Experiment 6	264
37. Component Percent Utilization For Select Queries	266
38. Component Percent Utilization For Insert Queries	267
39. Component Percent Utilization For Join Queries .	267
40. Component Percent Utilization For Difference Queries	268
41. Component Percent Utilization for Union Queries .	268

42. Query Mix Scenarios And Results For Experiment 7	270
43. Response Time As Number Of RCs And Relation Cardinality Are Increased Proportionally	273
44. Select Response Time As Disk Drives Are Added	275
45. IBM 3330 Disk Parameters	282
46. Processor Parameters	283
47. Execution Time For Selection On R With No Index	287
48. Execution Time For Selection On R With Index	287
49. Execution Time For Selection On R With No Index And Variable Number Of Processing Units	289
50. Execution Time For Selection On R With Index And Variable Number Of Processing Units	289
51. Execution Time For Joins On R And S	294
52. Execution Time For Joins On R And S For Variable Number Of Processing Units And Fixed Join Selectivity Factor	294
53. Execution Time For Aggregate Function Queries For Variable Number Of Partitions Of R	302
54. Execution Time For Aggregate Function Queries For Variable Number Of Processing Units	303

LIST OF FIGURES

1. The Taxonomy Of Database Systems	5
2. Conventional Database System	8
3. Distributed Database System	10
4. Single-Backend Database System	14
5. The DIRECT Architecture	17
6. DIRECT First Implementation	20
7. RDBM Architecture	22
8. Stonebraker's Machine Architecture	26
9. Teradata DBC/1012 Architecture	29
10. DBMAC Architecture	32
11. Parts and Models Database	40
12. The Select Query	42
13. The Project Query	43
14. The Insert Command	45
15. The Delete Query	46
16. The Update Command	47
17. Union, Difference And Intersect Commands	49
18. The Join Operation	50
19. Aggregate Operations	51
20. Five Phase RRDS Design Process	53
21. Generic Multi-Backend Architecture	61
22. Channel And Device Limitation	64
23. A Preliminary RRDS Architecture	66
24. Single-Queue Database Machine Model	71

25. Generic Multi-Backend System Queuing System Model	73
26. Relationship Between Events, Activities, Entities, And Processes	75
27. SLAM Network For Database Machine Context (Single-Queue) Model	77
28. Network Graph For S-Arch	82
29. Network Graph For M-Arch	89
30. Network Graph For F-Arch	99
31. Network Graph For P-Arch	106
32. Effect Of Degree Of Parallelism On S-Arch Response Time	116
33. Increased Database Size And Increased Number Of Backends - Percent Increase In Response Time . . .	121
34. Distribution Of EMPLOYEES Across A Two-RC System .	130
35. Clustering Data Structures For EMPLOYEES Relation	134
36. B+_Tree Structure	149
37. Hierarchical Indexing In RRDS	154
38. Effect Of Changing Relation Size	168
39. Increased Number Of Predicates - Response Time . .	170
40. Possible Outcomes Of Arbitrary Data Placement . .	176
41. Example Of Round Robin Data Placement	178
42. Partitioning Data Structures For EMPLOYEES Relation	184
43. Partitioning Of EMPLOYEES Relation Resulting From VRP Of Figure 42 And Assignment Of Partitions To The RCs	185
44. Effect Of Degree Of Parallelism And Response Set Size On Performance	191
45. Effect Of Increasing Number Of RCs	193

46. Response Time Versus Interarrival Time For Select Queries	204
47. Effect Of Response Set Size On Select Response Time	205
48. RRDS Simulation Model Approach Context	224
49. First Decomposition For Simulation Model	226
50. Query Execution Portion Of Simulation Model (2nd Decomposition)	228
51. DFD For Select Query Execution	230
52. Impact Of Different Parameters On Select Response Time	251
53. Impact Of Different Parameters On Insert Response Time	252
54. Impact Of Different Parameters On Join Response Time	253
55. Impact Of Different Parameters On Difference Response Time	254
56. Impact Of Different Parameters On Union Response Time	255
57. Effect Of IAT On Select Response Time	258
58. Effect Of IAT On Insert Response Time	259
59. Effect Of IAT On Join Response Time	260
60. Effect Of IAT On Difference Response Time	261
61. Effect Of IAT On Union Response Time	262
62. MDBM Architecture	281

CHAPTER 1

INTRODUCTION

With the recent decline in data processing hardware costs much of the research in the field of very large databases has been dedicated to exploring configurations characterized by multiple processing elements. Originally, the idea of dedicating a general-purpose computer (backend) to database management was explored (Canaday 1974; Cullinane 1975; Maryanski 1980). Studies generally indicate that this approach yields gains in performance and functionality only to a limited extent. In an effort to exploit the power of parallel processing, the concept of database machines emerged as an approach to improving access to very large databases (Copeland 1973; Su 1975; Ozkarahan 1975,1977; Lin 1976; Banerjee 1978; Leilich 1978; Schuster 1979). Unfortunately, many of these architectures rely on the commercial availability of mass storage technology as well as customized microcode and VLSI components. While the processor-per-track, processor-per-head, and off-the-disk designs (Haran 1983) that typify the database machine approach are based upon unique hardware configurations and technologies, a new software-oriented approach has emerged geared towards exploiting parallel processing while relying on conventional hardware. The relational replicated database system (RRDS) typifies this software approach. RRDS is a relational multi-backend system being developed to explore the possibility

of using multiple, commercially-available minicomputers and disk drives to achieve throughput gains and response time improvement. The goal in designing and implementing this system is to create a robust, extensible, database system realizable through conventional off-the-shelf hardware. The RRDS concept differs significantly from previously proposed systems in that it does not rely on any custom components and supports the relational data model with a complete data manipulation language (DML). Major design goals include: achieving performance proportional to the number of processing elements, creating a system which is commercially viable, and one which incorporates all of the capabilities of the relational model.

The RRDS project will ultimately result in the implementation and testing of a hardware configuration of multiple minicomputer systems and a software relational database system for the configuration. The portion of the RRDS project described in this dissertation includes the detailed design and preliminary performance analysis of the system architecture. Analytical techniques and simulations are used to make design decisions and predict system performance prior to prototyping.

Chapter 2 presents a survey of the literature to provide a context for the work. A taxonomy of existing database systems, along with a brief look at some existing relational multi-backend systems, provides a starting point for RRDS development. In Chapter 3 the development process is discussed in detail and a methodology presented for accomplishing the goals. In Chapter 4

a set of hardware design questions and a preliminary hardware configuration for RRDS are developed. Chapters 5 through 7 address software design questions and detail the data access strategies, data placement strategies, and directory management strategies developed for RRDS. The performance analysis and results are presented in Chapter 8, followed by a brief analytical comparison with some existing systems, in Chapter 9. Finally, a summary and suggestions for further research are given in Chapter 10.

CHAPTER 2

SURVEY OF THE LITERATURE

In this chapter existing database system architectures are classified and discussed. The taxonomy places database systems into three categories--conventional systems, backend systems, and distributed database systems. The backend system category is further decomposed according to whether systems are hardware based or software based. Presentation of the taxonomy along with a brief discussion of the merits and drawbacks of each approach puts the RRDS project into perspective and provides a context for the research. After discussing the various classes of systems briefly, the relational multi-backend architectures are presented in more detail. The design goals of the RRDS project, presented in Chapter 4, are intended to solve the problems inherent in these relational multi-backend architectures.

A Classification Of The Existing Database Systems

Database systems may be categorized according to the taxonomy of Figure 1. First, all database systems can be classified as conventional, backend, or distributed. The backend database systems are further subdivided into those which are based upon a hardware orientation and those which are based upon a software orientation.

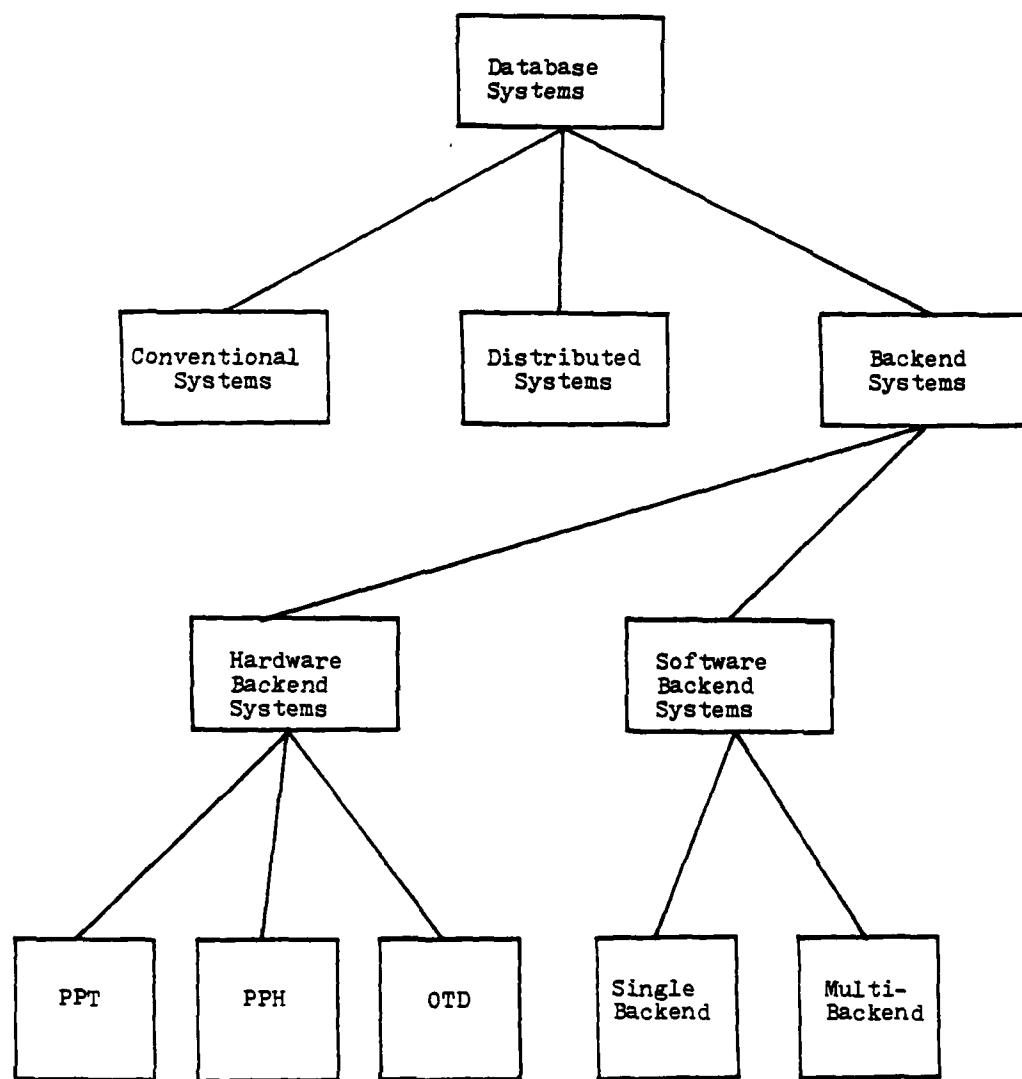


Figure 1. The Taxonomy Of Database Systems

Conventional database systems are the common, monolithic architectures. Distributed database systems are those featuring a collection of autonomous, geographically-dispersed systems which communicate over a long-haul communication network. The backend system concept evolved from a desire to relieve the host processor from the DBMS activity by offloading some, or all, of the DBMS functions to some backend system. This frees host resources, making them available for other activities not directly connected with the database. These backend architectures are broken down into two categories, depending on whether the processing elements comprising the backend system are hardware based or software based. The hardware-based systems are those which utilize specialized hardware to accomplish the majority of the database management functions. These are the architectures more commonly referred to as "database machines" and they normally employ multiple backend processors. On the other hand, software-based backend systems are those where specialized hardware is nonexistent or held to a minimum. The DBMS functions are accomplished in software in the backend network which can consist of one, or more, processors. In the following sections each of these categories of systems will be discussed with examples.

Conventional Database Systems

A conventional database system is contained in a single computer called a host. The DBMS software runs on the host and is managed by the host's operating system. The database is

stored on the secondary storage devices dedicated to the host processor. Limitations of this type architecture, illustrated in Figure 2, include reduced capacity, less reliability/availability, and the fact that the DBMS functions must contend with other applications for the host resources (Maryanski 1980). Performance upgrades in conventional database systems can be costly and disruptive, requiring replacement of expensive hardware or modification of software. In short, such systems are not extensible where extensibility of a database management system (Hsiao 1981a) is defined as the capability of the system for upgrade with:

- 1) no modification of existing software,
- 2) no additional programming,
- 3) no modification of existing hardware, and
- 4) no major disruption of system activity when additional hardware is being added.

Examples of conventional database management systems are INGRES (Stonebraker 1976b) and System R (Astrahan 1976).

Distributed Database Systems

Database systems with no central controller are those we generally call distributed database systems (DDBS). A distributed database is one which is not stored in its entirety at a single physical location, but rather is spread across a network of geographically-dispersed locations and connected via communication links (Date 1983). In general, a DDBS consists of a collection of sites, or nodes, connected together through a communication network where each site, in turn, constitutes an

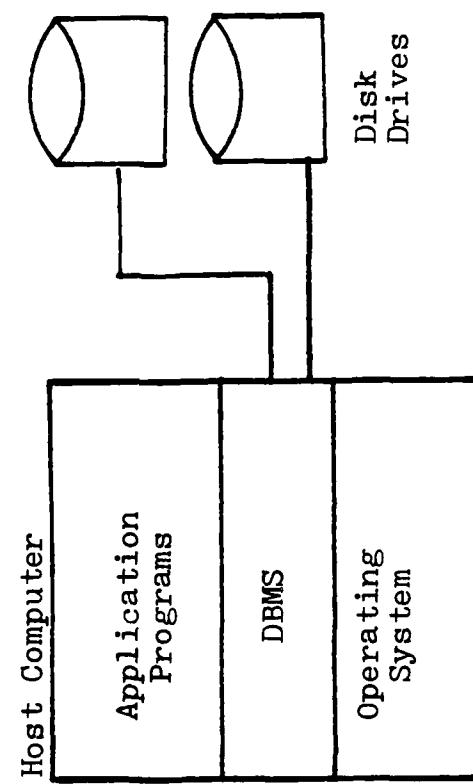


Figure 2. Conventional Database System

autonomous database system. Figure 3 illustrates this approach. Each site has its own database, and a processor running its own local DBMS. A DDBS may be either heterogeneous (as in Figure 3) or homogeneous, and the database may be replicated, partitioned, or a combination of the two. Often the expense of large data transfers and the need to locate data where it is actually processed requires duplicate databases.

Advantages of this approach include local autonomy, capacity and incremental growth, increased reliability and availability, and flexibility (Date 1983). Disadvantages include the need to duplicate databases, and complex concurrency control and security algorithms, which require large numbers of expensive control messages to be passed across the communication network (Date 1983).

Examples of distributed database systems include SDD-1 (Rothnie 1980), Distributed INGRES (Stonebraker 1976a), R* (Williams 1981), and MUFFIN (Stonebraker 1979). MUFFIN, which stands for Multiple Fast or Faster INGRES, is interesting because the architecture actually combines characteristics of DDBSs with the backend approach. A MUFFIN system consists of a number of "pods" interconnected via a low speed communication network. Each pod consists of a set of "A-Cells" and "D-Cells", interconnected via a high speed local network. A-Cells are conventional machines running application programs, operating systems, and Distributed INGRES software. D-Cells are processors dedicated to database management and running a partial version of

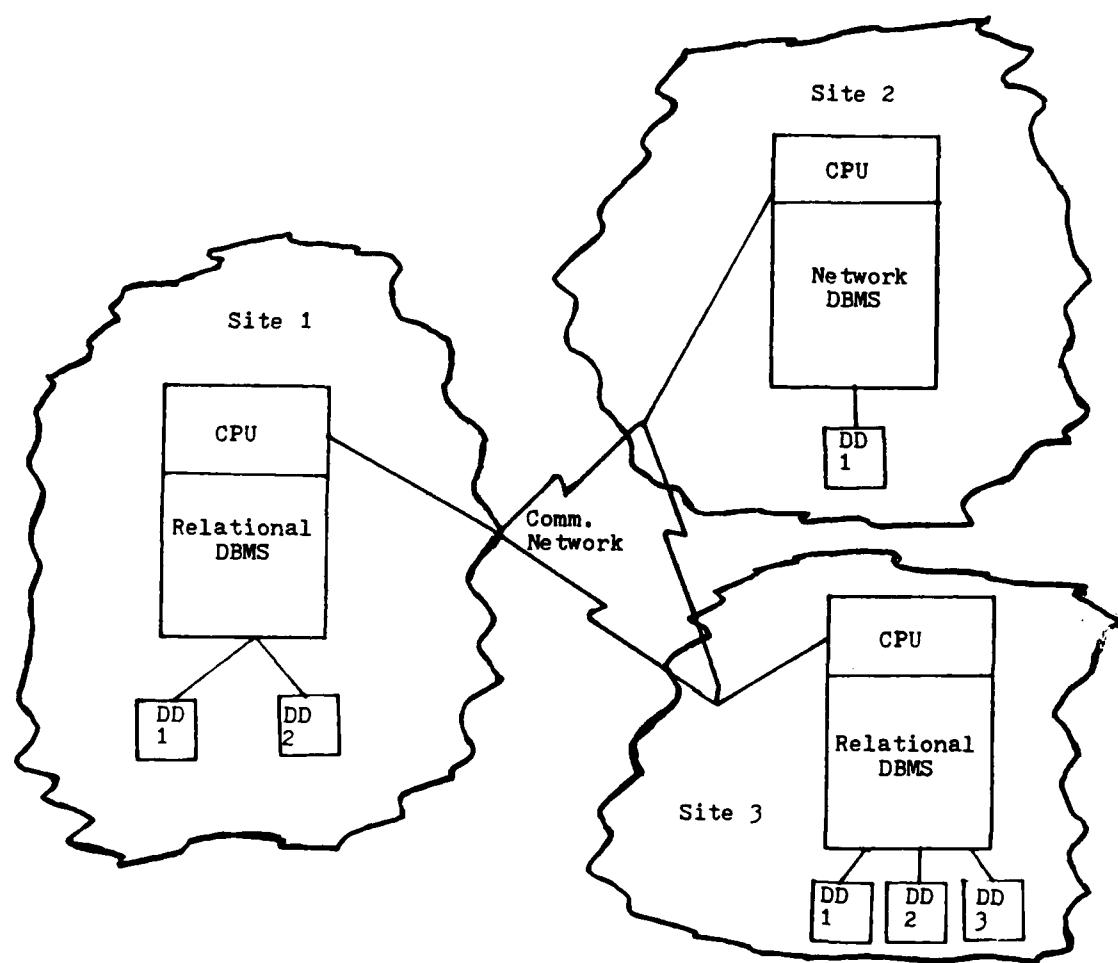


Figure 3. Distributed Database System

INGRES. They may therefore be viewed as backend processors in the network.

Though an area of continuing important research, distributed database systems are not the subject of this research.

Backend Database Systems

The third class of database systems shown in Figure 1 contains those systems known as backend database systems. The limitations of conventional database systems and the sheer size of today's databases led to the notion of offloading the DBMS functions onto separate, dedicated processors (backends). One notable feature common to all of the backend architectures is that they utilize some type of central controller. These architectures are further divided into those based upon a hardware approach and those based upon a software approach, to database management.

Hardware-Based Backend Systems

The hardware-based database systems, also called database machines, rely on special-purpose hardware to perform a large number of the DBMS functions. There are basically three approaches to these architectures (Haran 1983), sometimes called associative disk devices:

- 1) Processor-Per-Track (PPT),
- 2) Processor-Per-Head (PPH), and
- 3) Off-The-Disk (OTD).

These systems process the data "on-the-fly" while it is read from the disks. Special-purpose processors, which are associated with the secondary storage devices, are utilized to perform this processing.

PPT devices, also called cellular logic devices, may be regarded as an upgraded form of fixed-head disk (Date 1983). Each track has its own dedicated processor and all processors can perform the same search operation in parallel, enabling the entire disk to be searched in one revolution. The Content Addressed Segment Sequential Memory (CASSM) (Su 1975), from the University of Florida, and the Relational Associative Processor (RAP) (Ozkaranhan 1975), from the University of Toronto, are examples of PPT implementations.

PPH devices employ one processor per surface of the disk, hence the amount of data which can be processed on-the-fly during one revolution is one track per surface (i.e., one cylinder). Moving the processors between cylinders requires a seek operation. The PPH approach may be viewed as an upgraded form of the moving head disk (Date 1983). Examples of systems incorporating PPH approach include the Database Computer (DBC) (Banerjee 1978), developed at The Ohio State University, and the Content Addressable File Store (CAFS) (Mitchell 1976).

The OTD category (also called processor-per-disk) employs conventional moving head disks with a conventional disk controller but interposes a filtering processor between the disk

controller and the channel. This filtering processor applies search logic on-the-fly, eliminating unnecessary data. The OTD approach provides the functionality of the PPT and PPH systems at a lower cost because there is less custom hardware, however, at a cost in terms of performance (Date 1983). The Intelligent Database Machine (IDM) (Britton Lee 1980) is an example of the OTD database machines.

The fundamental reason for introducing a database machine is to improve system performance. However, it is not clear, in practice, if such performance is achieved, except for special cases such as catalog and index support and text retrieval applications. In addition, some of these systems rely on hardware which is either technologically unfeasible or generally unavailable. The desire to develop a system for general purpose use in the very large database realm, and using no specialized hardware, causes us to reject the hardware-based approach for RRDS.

Software-Based Backend Systems

Software-based systems are those which do not employ a significant amount of special-purpose hardware and where most of the functions of database management are accomplished in software. These type systems include both single-backend and multi-backend architectures.

The single-backend approach, shown in Figure 4, features a single processor dedicated to database management. Among the

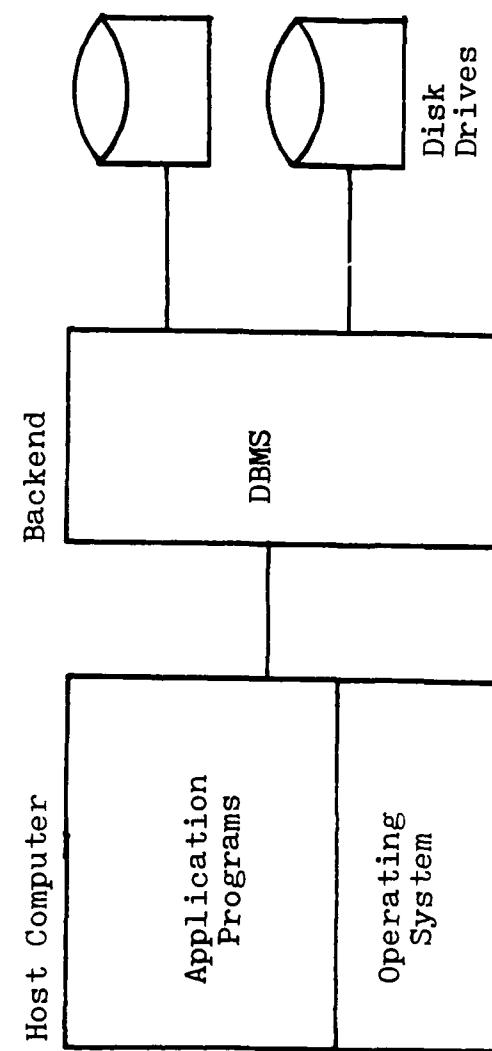


Figure 4. Single Backend Database System

advantages claimed of this type system are that performance upgrades are less disruptive, more manageable, and on a smaller scale than in a conventional system (Maryanski 1980). Upgrading the backend requires no modification of the application programs since they are executed in the host. Additionally, the backend separates characteristics of the secondary storage devices from those of the host. Hence, new storage technology can be incorporated without modifying the host. A major disadvantage of the single-backend design is that performance upgrades may require replacement of the backend possibly requiring software modification and hardware disruption. Furthermore, the performance upgrades can be achieved only to a limited extent. Therefore, these types of systems are still not fully extensible. Examples of single-backend database systems include XDMS (Canaday 1974) and General Electric's MADMAN machine (Maryanski 1980).

A desire for improved performance and extensibility led to the evolution of the multi-backend systems. They employ multiple computers for database management with a software-based DBMS and a controller. Since 1976 various architectures and approaches utilizing multiple processors have been proposed (Lowe 1976; DeWitt 1979; Stonebraker 1978; Hsiao 1981a, 1981b; Auer 1980; Miss 1980). Some do incorporate special-purpose hardware, and are not extensible. Various data models, including the relational and attribute-based models are also represented among the group. Regardless of the data model and the presence of some specialized hardware, all of the designs employ multiple

processors operating in some degree of parallelism (SIMD or MIMD) on the database. In addition, they all feature central controllers of various complexity and are software-based systems (i.e., the specialized hardware, if any exists, is not specifically responsible for the majority of the DBMS functions).

RRDS, the system proposed here, is a relational multi-backend concept aimed specifically toward extensibility and performance improvement. In the next section the advantages and disadvantages of five relational multi-backend architectures are discussed in detail. RRDS will be designed to overcome the shortcomings of these systems. A set of design questions will be formulated, in Chapter 3, for accomplishing this task.

Relational Multi-Backend Architectures

DIRECT

The relational database architecture known as DIRECT (DeWitt 1979), illustrated in Figure 5, grew from a desire for an MIMD implementation. The original proposal featured simultaneous execution of relational queries from different users as well as parallel processing of single queries. The architecture consists of four major components: a controller, a set of query processors (QPs), a set of charge coupled device (CCD) memory modules, and an interconnection matrix between the QPs and CCDs.

The backend controller (BEC) is responsible for interfacing the host with the QP network. It receives query packets from the

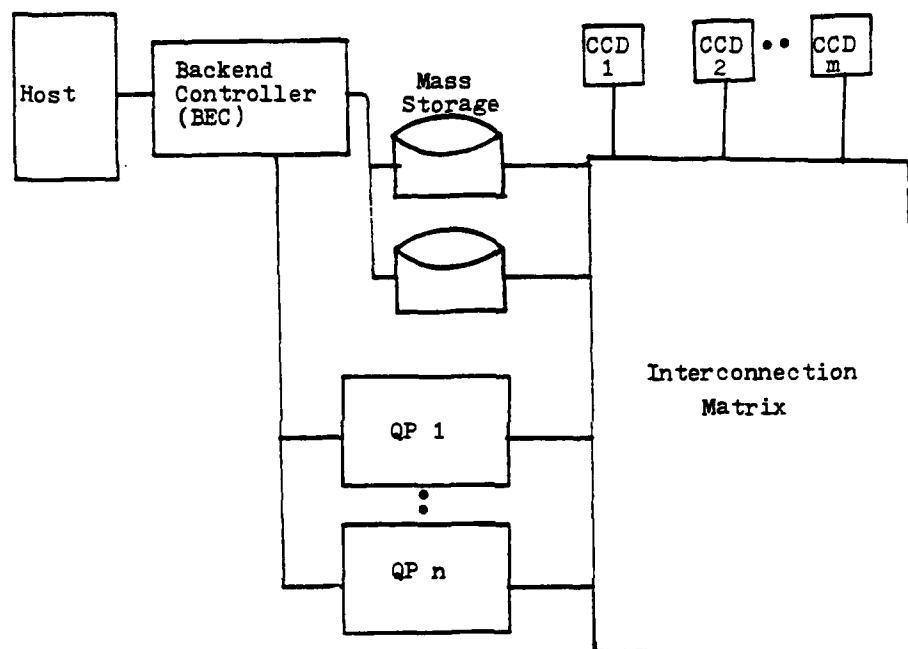


Figure 5. The DIRECT Architecture

host and determines the number of QPs which should be assigned to process the request. If the relations required by the request are not in the CCD memory, the controller pages them in from the mass storage prior to distributing the request to each of the QPs selected for its execution. The three main functions of the BEC are catalog management, instruction packet scheduling, and memory management.

The QPs are responsible for actually executing the requests they receive from the BEC and returning the results to the host via the BEC. DIRECT supports intraquery concurrency as well as interquery concurrency. Relations are divided into fixed-size pages and each QP, assigned by the BEC, associatively searches a subset of each relation referenced in the query packet. When a QP finishes examining one page of a relation it makes a request to the BEC for the address of the next page. The associative memory modules and interconnection matrix facilitate inter-query concurrency by permitting two QPs, each executing different queries, to search the same page of a common relation simultaneously. Each page frame of the associative memory is constructed from 8 CCD chips. A small page size (16K) results in more concurrency and less internal fragmentation, however, it does complicate the catalog management function of the controller.

The interconnection matrix must permit QPs to rapidly switch between page frames containing pages of the same or different relations. An expensive and complex cross-point switch is

required due to the bandwidth requirement for a large-scale implementation of DIRECT. In order to circumvent the expense of a traditional cross-point switch, the designers of DIRECT reversed the traditional roles of the processors and the memory. The CCD modules continuously broadcast their contents allowing any number of QPs to "listen" to the same memory element at the same time. This approach allowed the construction of a less complex switch, requiring no address lines, 1-bit wide data paths, and no conflict resolution hardware (DeWitt 1979). The drawback of this approach, however, is that such a crossbar switch is not suitable for a general purpose multiprocessor, making it a custom hardware component.

Due to the high cost of such custom hardware components, the first implementation of DIRECT featured a multi-port memory in place of the interconnection matrix and CCD memory modules. This architecture is illustrated in Figure 6. The multi-port memory, in response to a number of requests, time shares the data delivery among the QPs.

In addition to the fact that DIRECT, as proposed, required custom hardware, other problems are also present. The system suffers from controller limitation due to the fact that the BEC is required to completely supervise the processing of each query and perform directory management. The execution time of a query is dominated by the time required by the BEC to process messages, reducing the effective use of the parallel processing QPs. Also, much processing time is wasted moving data from the mass storage

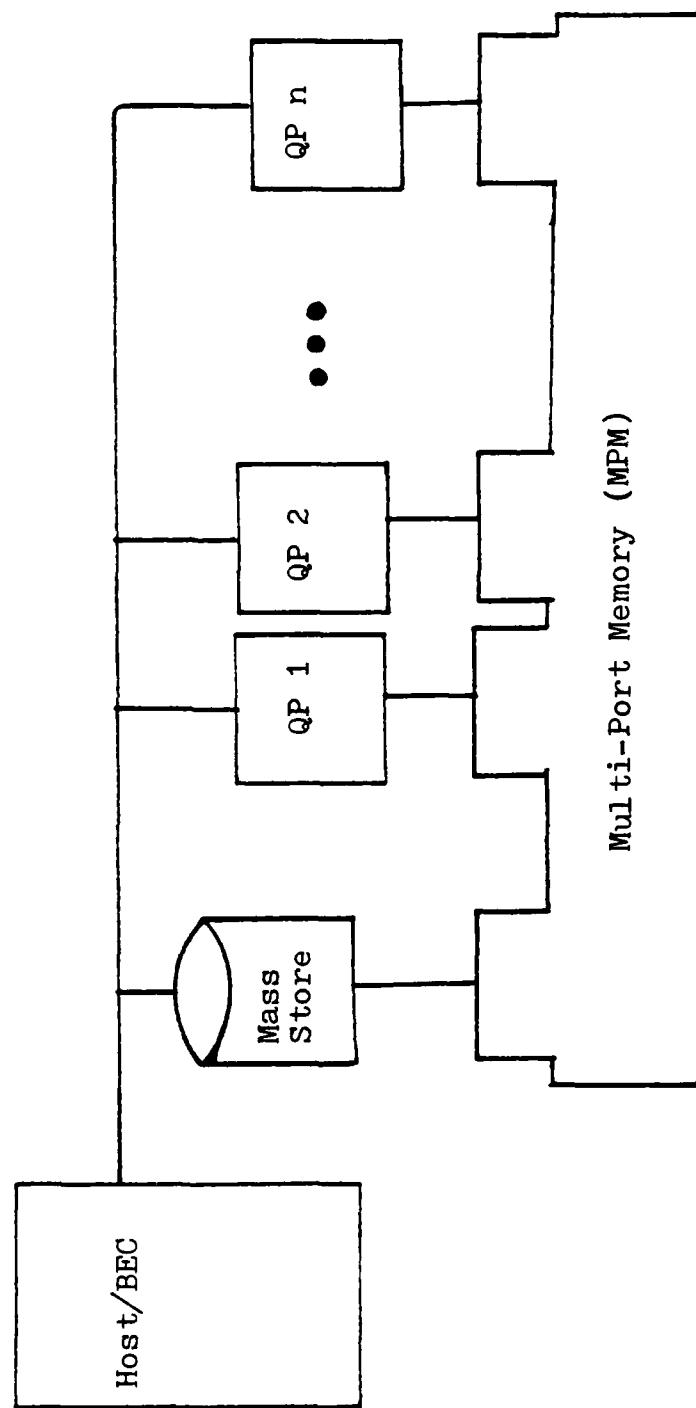


Figure 6. DIRECT First Implementation

to the CCD memory module (multi-port memory), and from there into the QP's memory before operations can be performed. A final problem found in DIRECT is the volume of control message traffic due to directory management. Each time a new page is required by a QP a message must be sent to the BEC and a message received from the BEC with the page's address. It has been estimated that about 8000 instructions are required (Boral 1981) to send a message from the BEC to a QP and vice versa. The architecture does not permit broadcasting of requests to the QPs from the BEC, and as a result a request which is to be executed by, say, three QPs will require three separate messages to be sent from the BEC to the QPs (approximately $8000 \times 3 = 24000$ instructions) taking up 24 milliseconds (msec) of controller time, assuming one instruction requires 1 msec (Hsiao 1981a).

RRDS design goals attempt to eliminate, or minimize, all of the disadvantages of DIRECT: hardware specialization, controller limitation, and control message traffic limitations.

Relational Database Machine (RDBM)

The Relational Database Machine (RDBM) (Auer 1980) has many of the features of a hardware-based architecture, but is included for discussion here because it employs a backend network dedicated to parallel processing of certain query types. The RDBM concept combines parallel processing and functional specialization in a unique multi-backend configuration (Figure 7). RDBM, a system of dedicated microprocessors supporting the

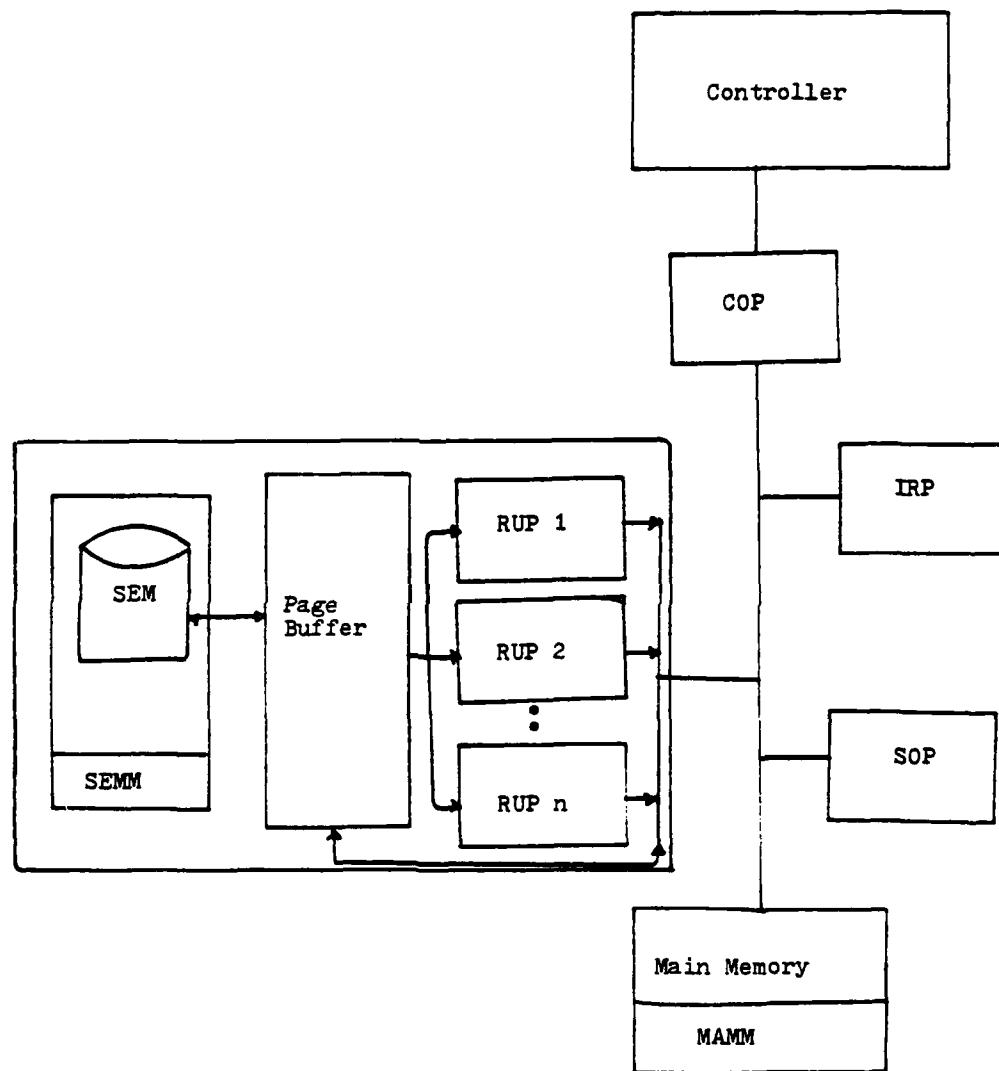


Figure 7. RDBM Architecture

main subtasks of mass storage operations and memory management, was designed primarily for efficiency. The objective was to design a complete architecture supporting the frequently used, time consuming, software DBMS functions with appropriate hardware components: a mass storage device with its own manager, a multiprocessor system consisting of special-purpose processors working on a large, common main memory, and a general-purpose minicomputer (controller) which controls system components and performs query preprocessing.

The mass storage device consists of conventional secondary memories (SEM), extended by a block buffer, the secondary memory manager (SEMM), and a number of processing elements called restriction and update processors (RUPs). The RUPs operate in parallel on different segments of a relation, performing the same query (Retrieve, Update, Insert, Delete). Functional specialization is realized in the multiprocessor system consisting of a sort processor (SOP), an interrecord processor (IRP), and a conversion processor (COP).

When the controller receives a query, it decomposes the query into elementary operations which are then performed partially in parallel. The controller is responsible for query analysis, optimization and code generation, and preparing the query for processing by various system components. Additionally, the controller supervises the actions of all the components via a bus system offering separate paths for data, instruction, and status transfers. Due to its heavy involvement in all phases of

query execution the controller becomes a limiting factor in system throughput.

Relational joins and record sorting are performed by the IRP and SOP, respectively. This functional specialization can result in an uneven distribution of workload. For example, in an environment where join queries predominate, the IRP will be overwhelmed and the SOP and RUP networks left mostly idle. Thus, the best utilization of the multiple processors is not being realized. In addition, loss of a processor (such as the IRP) results in complete loss of a capability, making such a system unreliable. This software specialization problem can be avoided if the functional approach is rejected in favor of one in which all the processors are capable of performing all of the DBMS functions.

Finally, the fact that RUPs can only examine records in the page buffer requires transfer of information from the secondary memory to the buffer, and from the main memory (MAM) to the buffer, prior to manipulation. The ultimate throughput limitation is the rate at which records can be read from the SEM to the RUPs via the interconnecting channel. Thus, after a certain point, adding RUPs will not improve system performance. A system which is limited due to interconnections among the secondary memory, processors, and main memory is said to suffer from channel limitation. Channel limitation should not exist in an extensible system. RRDS will be designed so that both the

software specialization and channel limitation problems, characteristic of RDBM, are not present.

Stonebraker's Machine

An architecture designed by Michael Stonebraker at the University of California at Berkeley (UCB), known simply as Stonebraker's machine (Hsiao 1981a), is a good example of a true software-based, relational multi-backend database system. A schematic of the system is shown in Figure 8. It consists of a central controller supervising the actions of a number of backend processors, each with one dedicated disk drive.

The controller preprocesses incoming queries, parsing and decomposing them into requests which access only a single relation. System directory information is stored at one of the backends designated as the "special backend". Following preprocessing, the controller consults the directory in the special backend, determining the other backends required to process the request. Queries are then forwarded from the controller to the appropriate backend(s) for processing. Each backend operates independently on the portion of the database stored in its disk drive and returns the results to the controller. The controller finally outputs the results to the user who issued the query.

This system, which is a large step in the right direction, still suffers many shortcomings. First, unlike DIRECT and RDBM, a query must be executed by one or more backends determined by

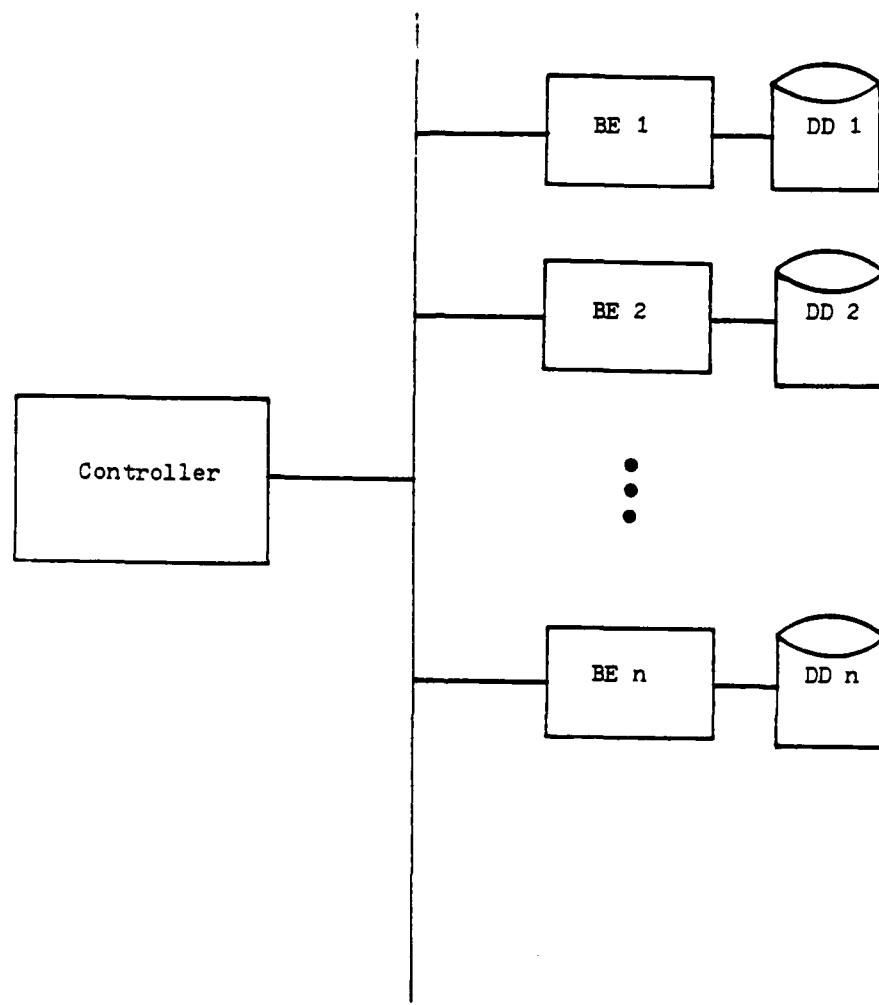


Figure 8. Stonebraker's Machine Architecture

the controller. For example, if a query requires information stored in the disk drive dedicated to the first backend this processor will be the only one involved in request execution. This approach does not fully utilize parallel processing on every query, and in the event of a series of queries requiring the same backend, serious bottleneck problems can occur quickly. In addition to this backend limitation problem (Hsiao 1981a) Stonebraker's machine also utilizes a specialized backend for all directory processing. Each query requires controller access to the special backend to determine which processors will participate in execution. Not only does this create a bottleneck at the special backend, it renders the system unreliable and greatly increases the amount of control message traffic. In an extensible system specialized components must be avoided at all cost. The directory management functions should, as much as possible, be accomplished in parallel by all the backends, just as the other DBMS functions.

The fact that each backend has only one dedicated disk drive also reduces extensibility and limits the ultimate capacity of the system. Addition of multiple disk drives to each backend would alleviate this device limitation problem and allow the system to better accommodate very large databases. Also, the data placement and access strategies of Stonebraker's machine require that an entire relation be retrieved in order to process a query. As a result, time is wasted retrieving all the records

in a relation when a response may contain only a portion of the relation.

A final drawback of this system is the lack of a broadcast capability. Queries requiring multiple backends cause messages to be sent to, and returned from, the special backend for directory management. Then, separate messages must be dispatched to each backend involved in the query execution. In order to reduce the volume of control message traffic and free the controller, a broadcast capability should be incorporated. An extensible design will completely, or partially, eliminate the limitations and problems manifest in Stonebraker's machine: backend limitation, specialized backends, control message traffic, controller limitation, and device limitation.

Teradata DBC/1012

The DBC/1012 (Teradata 1986), a commercially available database system, is designed to support very large database applications and to provide a degree of fault tolerance and extensibility. The system combines some special-purpose hardware with a software orientation to achieve these goals. A small DBC/1012 configuration is illustrated in Figure 9. The system represents a parallel processing architecture and it consists of controllers, called interface processors (IFPs), backends, called access module processors (AMPs), and secondary storage devices, called disk storage units (DSUs). The heart of the DBC/1012 consists of proprietary specialized hardware, called a YNet,

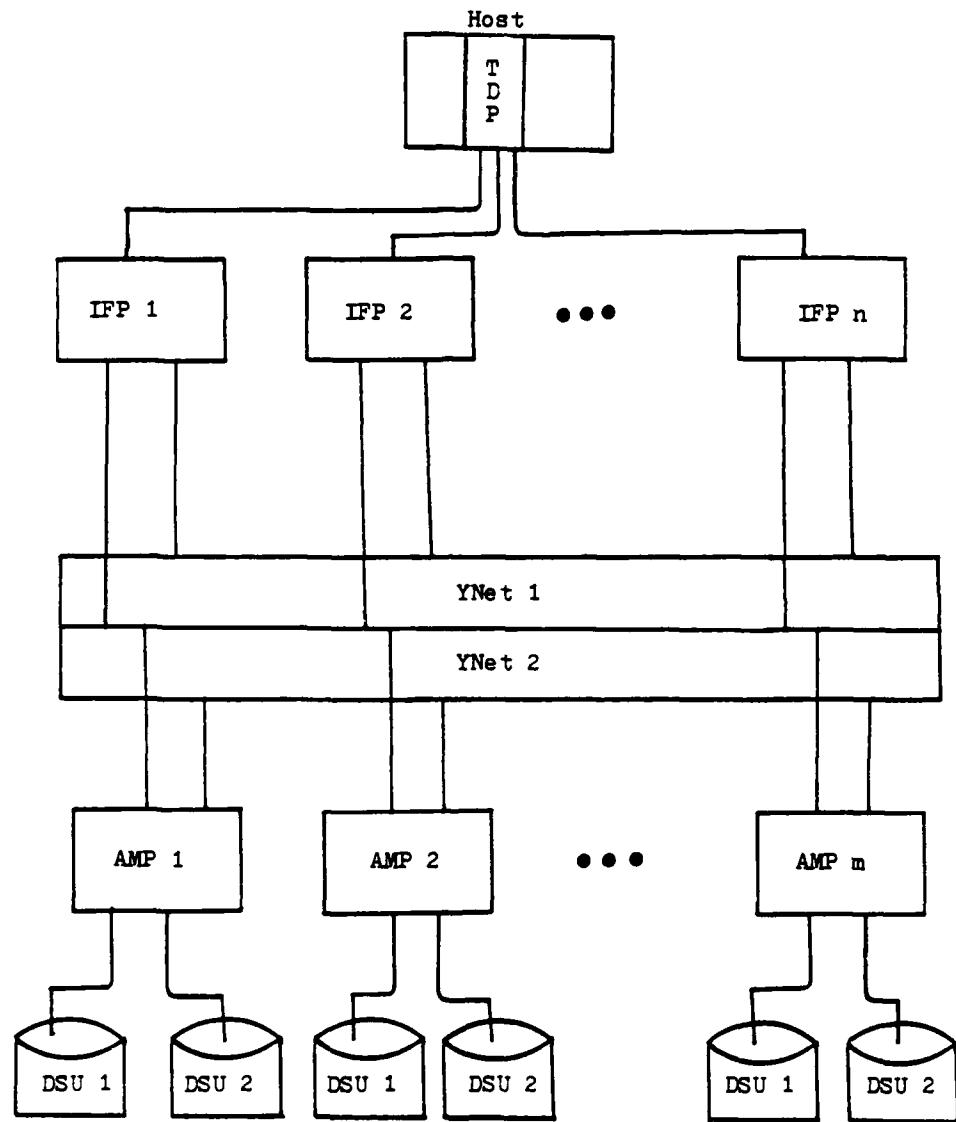


Figure 9. Teradata DBC/1012 Architecture

which interconnects the system (much like the interconnection network of DIRECT) and facilitates parallel processing. The YNet is actually a redundant active array of high speed logic with intelligence. The logic of the YNet provides selection and sorting functions for the system much like the filtering processors of some database machines.

A DBC/1012 can consist of between 3 and 1024 AMPs and must have at least two IFPs. A maximum of two DSUs may be dedicated to each AMP. The IFPs receive queries from the Teradata directory program (TDP) running on the host. After receiving a query, an IFP translates the query into a language suitable for the YNet/AMP network. Using the active directory the IFP then performs directory management functions and determines the work steps necessary to process the request and dispatches the instructions over the YNet to one or more AMPs. The AMPs accept the rudimentary instructions and perform the indicated processing on the portion of the database in their DSUs, returning the results to the IFPs via the YNet. Finally, under the supervision of the IFPs, the collated results (collation accomplished by the YNet) are returned to the host.

Though the DBC/1012 employs a network of controllers (a minimum of two) and utilizes proprietary special-purpose hardware, it does incorporate some features of interest. The fact that the relational database is distributed evenly across the DSU network and that each AMP performs the same DBMS functions make the DBC/1012 the most extensible system discussed

so far. The DBC/1012 does, however, exhibit two of the major problems previously discussed. First, there is a controller limitation problem due to the fact that directory management is accomplished by the controllers (IFPs). The performance of the system will be limited by the speed of the controllers as well as the number of controllers. In addition, as the size of the controller network grows communication and coordination with this network presents a problem. Second, the DBC/1012 exhibits the device limitation problem since each AMP can accommodate a maximum of two DSUs. In order to achieve capacity growth, a large number of AMP/DSU units must be added to the YNet, increasing the workload of the controller network and further aggravating the controller limitation problem.

DBMAC

The Italian database system, known as DBMAC (Missikoff 1980; Cesarine 1983) also incorporates special-purpose filtering hardware in a primarily software-based architecture. The system, illustrated in Figure 10, consists of a global memory which communicates with a network of processing units (PUs) via a global bus (G-bus). The PU network communicates with a secondary storage network via a second bus called a mass memory bus (MM-Bus). The secondary storage network consists of a set of intelligent memory interfaces (IMIs) connected to disk drives. The IMI units are actually filters for performing selection operations on-the-fly as records are read off the disks (as in the OTD architectures discussed in Section 2.4.1).

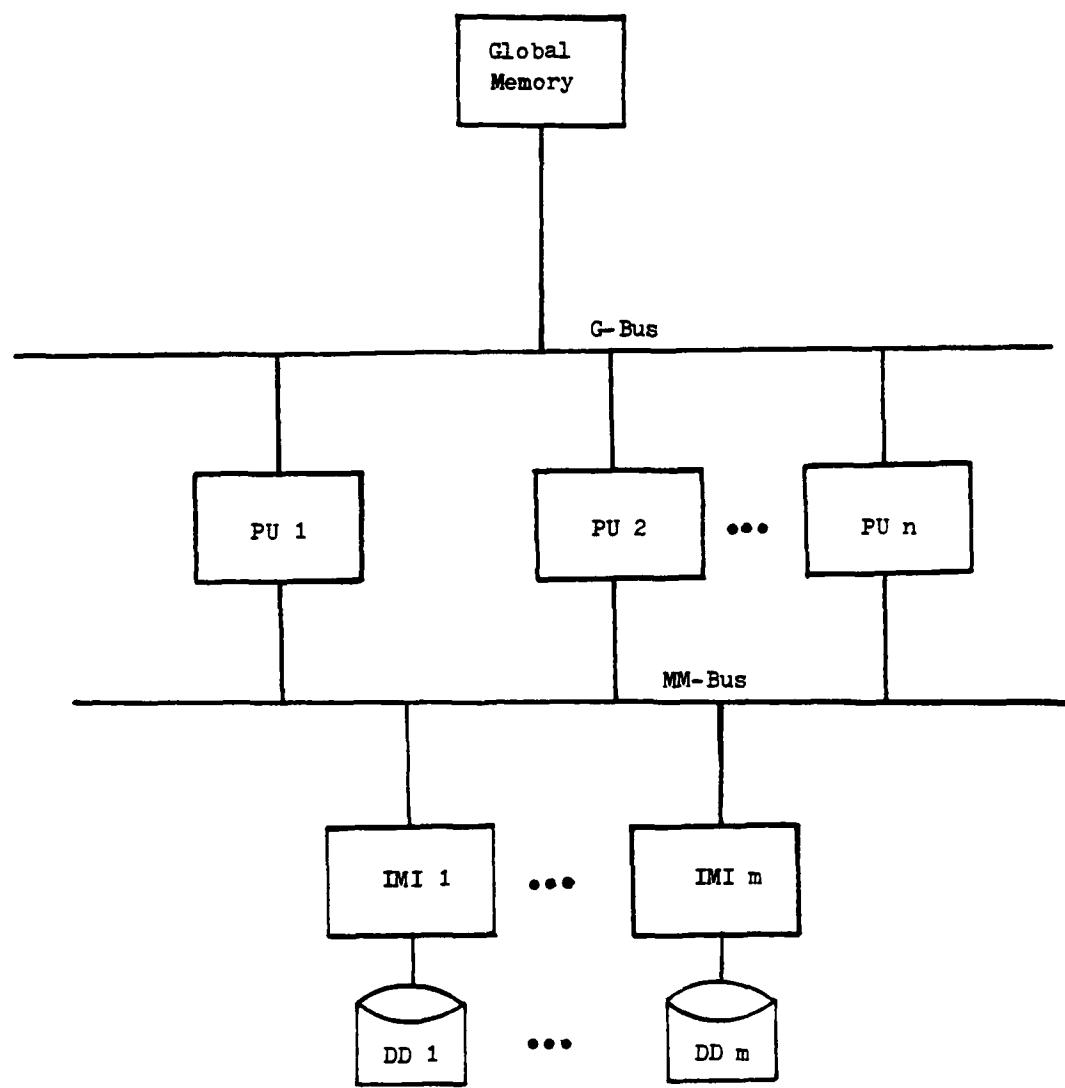


Figure 10. DBMAC Architecture

DBMAC can be considered a software-based system because most of the DBMS functions are implemented in software in the PU network. The most interesting aspect of the system is the manner in which these functions are implemented and the obvious lack of a controller component. Though there is no controller illustrated in Figure 10, one nevertheless exists--within the PU network. The controller functions and all of the DBMS functions, not accomplished by the IMI units, are performed by a set of modules distributed across the PUs. The set of modules of a task (e.g., request preprocessing and concurrency control) are placed such that, as much as possible, no two modules of the task are resident in the same PU. Therefore, all the system tasks are executed by the PUs in a distributed fashion. Communication among the PUs is via the G-bus.

Though an interesting approach to multi-backend design, this system is not extensible and has many drawbacks. First, the secondary storage devices are not dedicated to the PUs. The system exhibits the channel limitation problem and its throughput is limited by the speed of the mass bus attached to the secondary storage network (Hsiao 1981a). This occurs because, even though there are multiple PUs, they cannot access different portions of the secondary storage simultaneously. In addition, throughput will be limited by the speed of the G-bus as all of the system tasks, distributed over the PU network, attempt to coordinate their actions through this bus. A major problem with DBMAC, which makes it nonextensible, is software specialization. Since

each PU contains a separate module from each system task the software in the PUs is not identical. This leads to a decrease in system reliability, even more severe than the functional specialization approach of RDBM, since the loss of one PU will render the system incapable of performing any DBMS function. Extensibility is lost because addition of new PUs will require the redistribution of the controller and DBMS software modules.

This concludes the discussion of existing software-based architectures. Based upon the limitations and problems discussed here, a set of design considerations for developing RRDS will be formulated in the next chapter. The methodology used for developing RRDS is also presented in Chapter 3.

CHAPTER 3

THE RRDS DESIGN PROCESS

This chapter describes the methodology used for developing the RRDS detailed design. First, based upon limitations of systems described in Chapter 2, the project goals are enumerated and a corresponding set of hardware design questions formulated. Next, an appropriate data model is chosen for RRDS and briefly described. The set of operations to be supported and a complete data manipulation language (DML) are also provided. Given the design goals and the underlying data model a methodology is presented for developing a generic software multi-backend architecture from which a preliminary RRDS configuration will evolve. Once the preliminary hardware configuration is identified a set of software design questions will be formulated addressing issues such as data access, data placement, and directory management strategies. Finally, a performance analysis plan for RRDS will be presented.

Research Goals

The goal of this research effort is to investigate the possibility of using a multiplicity of general-purpose processors and storage elements, novel hardware configuration and innovative software structure to achieve throughput gain and response time improvement over conventional database management systems. This

investigation will result in the design of a robust and extensible database system. RRDS differs from previously proposed systems in either the hardware organization, the software structure, or the data model. The system will not rely on any custom components and will support the relational data model with a complete DML. The goal of achieving performance proportional to the number of processing elements will be paramount, as well as the desire to create a system which is commercially viable and incorporates all the capabilities of the relational model.

The RRDS project will ultimately result in the detailed design and implementation of a relational database system to support very large databases. The objective of this research effort is to accomplish the detailed design and performance analysis, producing the complete RRDS architecture. RRDS should conform to the design goals of Table 1. In the next section a set of hardware design questions is formulated which will lead to a preliminary RRDS architecture.

Hardware Architectural Design Considerations

In Chapter 2 five relational multi-backend database systems were described. For each system the advantages and disadvantages of the architecture, as well as potential bottlenecks, were identified. Table 2 presents the six hardware design questions formulated as a result of these findings. In addition to satisfying the project goals presented earlier, the RRDS design

TABLE 1. RRDS DESIGN GOALS

- The system must be extensible
- The system performance must be proportional to the number of processing elements
- Maximum use of parallel processing, for improved throughput, should be made
- The system must support the entire relational data model
- The system must rely on off-the-shelf hardware

will strive to eliminate, or minimize, these adverse conditions.

Selecting a Data Model and a Data Manipulation Language

RRDS, designed specifically for very large database applications, is based upon the relational data model. This model is both simple and elegant, with all the information in the database, including entities and relationships, represented in a simple, uniform manner--the relation. This uniformity of data representation leads to a uniformity in the operator set, i.e., only one of each type operator is required for the basic data manipulation functions. The relational model facilitates the development of a high-level data manipulation language (DML) dealing with entire sets as operands. One of the major strengths of this approach is that languages such as the relational algebra and relational calculus may be simply defined, proven complete,

TABLE 2. MULTI-BACKEND DATABASE SYSTEM HARDWARE DESIGN QUESTIONS

DESIGN QUESTION	ADVERSE EFFECT	RESULT
1. Controller Limitation	Bottleneck	Limited throughput due to poor response time
2. Comm Net Limitation	Bottleneck	
a) Too many messages		Restricted growth due to performance anomaly
b) Messages too large		Limited throughput due to poor response time
3. Backend Specialization	Bottleneck Loss of Database Function	Limited throughput Limited availability
4. Backend Limitation	Bottleneck	Limited throughput due to loss of parallel processing
5. Channel Limitation	Bottleneck	Limited throughput
6. Device Limitation	Less Capacity	System accommodating very large databases may be too expensive

and are still extremely powerful. This section details the high-level DML developed for RRDS. The language incorporates all of the capabilities of a complete relational language.

In the following paragraphs, all of the basic RRDS operations are introduced and illustrated with examples. Appendix A contains the formal BNF specification for the DML. The RRDS DML supports the complete set of operations available in the relational algebra, including one-relation operations and two-relation operations. In addition, RRDS provides the following aggregate capabilities: count, sum, min, max, and average. The RRDS operations are listed in Table 3. In what follows a brief description of each of these operations is provided along with examples based upon the simple parts-models database of Figure 11. The Parts relation contains information about electrical components which are used in the manufacturing of two appliances, Model1 and Model2. Each model relation contains the parts required to assemble that model as well as the quantity required.

TABLE 3. THE RRDS OPERATIONS

One-Relation Operations	Two-Relation Operations
Select Project Insert Delete Update Aggregates	Union Difference Intersection Join

Database Schema:

Parts (Part#, Color, Type)

Modell (Part#, Qty)

Model2 (Part#, Qty)

Sample Database:

Parts:

Part#	Color	Type
T21	Red	Transistor
T22	Red	Transistor
R37	Green	Resistor
R16	Green	Resistor
C26	Grey	Chassis
D47	White	Diode
L1	Red	LED
L2	Green	LED
L3	Amber	LED

Modell:

Part#	Qty
T21	60
C26	1
L1	8
L3	8
R37	20

Model2:

Part#	Qty
T22	30
R16	20
D47	10
L1	5
L2	5
L3	10
C26	1

Figure 11. Parts and Models Database

A query in RRDS is a request for data manipulation involving one of the operations from Table 3, which may or may not be qualified by a set of specifications. A specification is a conjunction of predicates, a predicate being of the form: (attribute relational-operator value). Relational operators are from the set: $\{=, <, >, >=, <, <=\}$. An example predicate is: (Color = Red), and a conjunction of predicates which might be used to qualify a query is: (Color = Red) AND (Type = Transistor).

The SELECT operation is used to obtain a specified set of records from a relation. The selection may be qualified or unqualified. An unqualified selection, identified by the reserved word ALL, returns all of the records of a relation. A qualified selection query returns the set of records from the target relation satisfying the specification. Examples of the select query are illustrated in Figure 12.

The PROJECT operation yields a vertical subset of a specified relation, i.e., the unique values obtained by selecting specified attributes in a relation. As in the selection query, projections may also be qualified or unqualified. Examples of the project query are shown in Figure 13.

The INSERT operation is used to add a new record to the database. Each insert command causes one record to be inserted into the target relation. In addition to the target relation,

Query Format: `SELECT [ALL] FROM <Target-Relation>
[WHERE
<specifier>]`

Example 1: Unqualified SELECT:

Query:
`SELECT ALL FROM Parts`

Result:

Part#	Color	Type
T21	Red	Transistor
T22	Red	Transistor
R37	Green	Resistor
R16	Green	Resistor
C26	Grey	Chassis
D47	White	Diode
L1	Red	LED
L2	Green	LED
L3	Amber	LED

Example 2: Qualified SELECT:

Query:
`SELECT FROM Parts
WHERE
((Color = Red)
AND
(Type = Transistor))`

Result:

Part#	Color	Type
T21	Red	Transistor
T22	Red	Transistor

Figure 12. The Select Query

Query Format: PROJECT (<Attr-List>) FROM <Target-Relation>
[WHERE
<specifier>)]

Example 1: Unqualified Projection:

Query:

PROJECT (Part#) FROM Model1

Result:

PART#
T21
C26
L1
L3
R37

Example 2: Qualified Projection:

Query:

PROJECT (Part#) FROM Model1
WHERE
(Qty > 10)

Result:

PART#
T21
R37

Figure 13. Project Query

this command also specifies the values to be inserted into the relation's attribute fields. The insert command is illustrated in Figure 14.

Records are removed from the RRDS database by the issuance of a DELETE command. All records satisfying the specification are deleted from the specified target relation. The reserved word ALL is used when one wishes to delete all records, in a relation, with one command. The format and examples of the delete command are illustrated in Figure 15.

The UPDATE operation is used to change values in records already in the RRDS database. An unqualified update command results in the modification of all records in the target relation. A subset of the records in the target relation can be modified by using an update command with a qualification (specifier). The new values for the attributes being modified are specified by the assignment statement list. For example, if an employer wishes to grant his employees a \$5,000 raise, the assignment statement: (Salary := Salary + 5000), where Salary is an attribute in the target relation, would be specified in the issued update command. The format and examples of update commands are illustrated in Figure 16.

The UNION, DIFFERENCE, and INTERSECTION operations all require two relations as operands. The result is produced by the set operation specified. All three of these operations require the target relations to have identical schemas, such as the

Query Format: INSERT (<Record>) INTO <Target-Relation>

Example:

Query:

INSERT (Part# = L2,Qty = 15) INTO Modell

Result:

Modell:

PART#	QTY
T21	60
C26	1
L1	8
L3	8
R37	20
L2	15

Figure 14. The Insert Command

Query Format: DELETE [ALL] FROM <Target-Relation>
[WHERE
(<specifier>)]

Example 1: Delete All:

Query:	Result:						
DELETE ALL FROM Parts	Parts:						
	<table border="1"><thead><tr><th>PART#</th><th>COLOR</th><th>TYPE</th></tr></thead><tbody><tr><td></td><td></td><td></td></tr></tbody></table>	PART#	COLOR	TYPE			
PART#	COLOR	TYPE					

Example 2: Qualified Delete:

Query:	Result:																					
DELETE FROM Parts WHERE (Color = Red)	Parts:																					
	<table border="1"><thead><tr><th>PART#</th><th>COLOR</th><th>TYPE</th></tr></thead><tbody><tr><td>R37</td><td>Green</td><td>Resistor</td></tr><tr><td>R16</td><td>Green</td><td>Resistor</td></tr><tr><td>C26</td><td>Grey</td><td>Chassis</td></tr><tr><td>D47</td><td>White</td><td>Diode</td></tr><tr><td>L2</td><td>Green</td><td>LED</td></tr><tr><td>L3</td><td>Amber</td><td>LED</td></tr></tbody></table>	PART#	COLOR	TYPE	R37	Green	Resistor	R16	Green	Resistor	C26	Grey	Chassis	D47	White	Diode	L2	Green	LED	L3	Amber	LED
PART#	COLOR	TYPE																				
R37	Green	Resistor																				
R16	Green	Resistor																				
C26	Grey	Chassis																				
D47	White	Diode																				
L2	Green	LED																				
L3	Amber	LED																				

Figure 15. The Delete Query

Query Format: UPDATE (<Modifier>) IN <Target-Relation>
 [WHERE
 (<specifier>)]

Example 1: Unqualified Update:

Query:

UPDATE (Qty := Qty + 2) IN Model2

Result: Model2

PART#	QTY
T22	32
R16	22
D47	12
L1	7
L2	7
L3	12
C26	3

Example 2: Qualified Update:

Query:

UPDATE (Color := Blue) IN Parts
 WHERE
 (Type = Transistor)

Result: Parts

PART#	COLOR	TYPE
T21	Blue	Transistor
T22	Blue	Transistor
R37	Green	Resistor
R16	Green	Resistor
C26	Grey	Chassis
D47	White	Diode
L1	Red	LED
L2	Green	LED
L3	Amber	LED

Example 3: Qualified Update:

Query:

UPDATE (Color:=Blue, Type:=Cboard)
 IN Parts
 WHERE
 (PART# = C26)

Result: Parts

PART#	COLOR	TYPE
T21	Red	Transistor
T22	Red	Transistor
R37	Green	Resistor
R16	Green	Resistor
C26	Blue	Cboard
D47	White	Diode
L1	Red	LED
L2	Green	LED
L3	Amber	LED

Figure 16. The Update Command

relations Model1 and Model2 of our parts-models database. The union command results in the combination of the records of the two target relations into one relation (duplicate tuples are eliminated). When a difference command is issued the records which exist in the first target relation, but not in the second target relation, are returned. Finally, the intersection command returns those records contained in both target relations. All of these two-relation operations are illustrated in Figure 17.

The final two-relation operation performed by RRDS is the JOIN operation. As in the queries of Figure 17, the join also takes entire relations as operands, however, they need not have identical schemas. Instead, they must have one or more attributes in common. We say the relations are joined over the common attribute(s). The result of a join operation, over some common attributes, is a new relation with all the attributes of the original target relations and the information from those records which have the same value for the common attributes. The format and an example of a join command are shown in Figure 18 (in order to better illustrate this operation we have added a fourth relation, Suppliers, to our Parts and Models database).

The final atomic operations performed by RRDS, for the purpose of statistical computations, are the aggregate operations. The following scalar aggregates are supported by the system: COUNT, SUM, MIN, MAX, and AVERAGE. The aggregate operations can have entire relations or parts of them as their operand. All of the aggregates are illustrated in Figure 19.

Query Formats:

<Target-Relation> UNION <Target-Relation>
 <Target-Relation> DIFF <Target-Relation>
 <Target-Relation> INTSECT <Target-Relation>

Example 1: Union:

Query:

Model1 UNION Model2

Result:

PART#	QTY
T21	60
T22	30
C26	1
R16	20
D47	10
L1	8
L1	5
L2	5
L3	8
L3	10
R37	20

Example 2: Difference:

Query:

Model1 DIFF Model2

Result:

PART#	QTY
T22	30
R16	20
D47	10
L1	5
L2	5
L3	10

Example 3: Intersection:

Query:

Model1 INTSECT Model2

Result:

PART#	QTY
C26	1

Figure 17. Union, Difference, and Intersect Commands

SUPPLIERS:

S#	PART#	CITY
S1	T21	NY
S1	T22	NY
S2	R37	LA
S2	R16	LA

Query Format: <Target-Relation> JOIN <Target-Relation>

Example: Join:

Query: Parts JOIN Suppliers

Result:

PART#	COLOR	TYPE	S#	CITY
T21	Red	Transistor	S1	NY
T22	Red	Transistor	S1	NY
R37	Green	Resistor	S2	LA
R16	Green	Resistor	S2	LA

Figure 18. The Join Operation

Aggregate Format/Explanation:

COUNT FROM <Target-Relation> [WHERE (<specifier>)]

Returns number of records in target relation (or number of records in a subset of the target relation).

SUM (<Attribute>) FROM <Target-Relation> [WHERE (<specifier>)]

Returns sum of values for an attribute in target relation (or in a subset of the target relation).

MIN (<Attribute>) FROM <Target-Relation> [WHERE (<specifier>)]

MAX (<Attribute>) FROM <Target-Relation> [WHERE (<specifier>)]

Returns the minimum/maximum value for an attribute in target relation (or in a subset of the target relation).

AVERAGE (<Attribute>) FROM <Target-Relation> [WHERE (<specifier>)]

Returns average of values for an attribute in target relation (or in a subset of the target relation).

Examples:

Aggregate Operation:	Result:
COUNT FROM Parts	9
SUM (Qty) FROM Model2	81
MIN (Qty) FROM Model2	1
MAX (Qty) FROM Model2	30
AVERAGE (Qty) FROM Model2	11.5

Figure 19. Aggregate Operations

In this section we have specified the DML for RRDS. This specification will be used to implement the DML compiler module of the RRDS controller. The language combines the power and flexibility of the relational algebra along with the statistical computation capability of aggregate operations. The next section describes the detailed design process which will be used to develop RRDS.

The Design Process

RRDS design will begin with the development of a generic software multi-backend database system. The generic architecture, along with the set of design goals and questions, will then be subjected to the five-phase, iterative design process illustrated in Figure 20. The complete data flow diagram (DFD) for the design methodology model is presented in Appendix B.

In phase one a preliminary hardware architecture will be developed from the generic configuration. Analytical models and simulations of the preliminary RRDS architecture and three other variations on the generic architecture will be compared with respect to the design questions of Table 2. In subsequent phases of development, software questions will be addressed. In each phase a major new capability will be added to the system. The software design questions are described briefly here and discussed in more detail in chapters 5, 6, and 7.

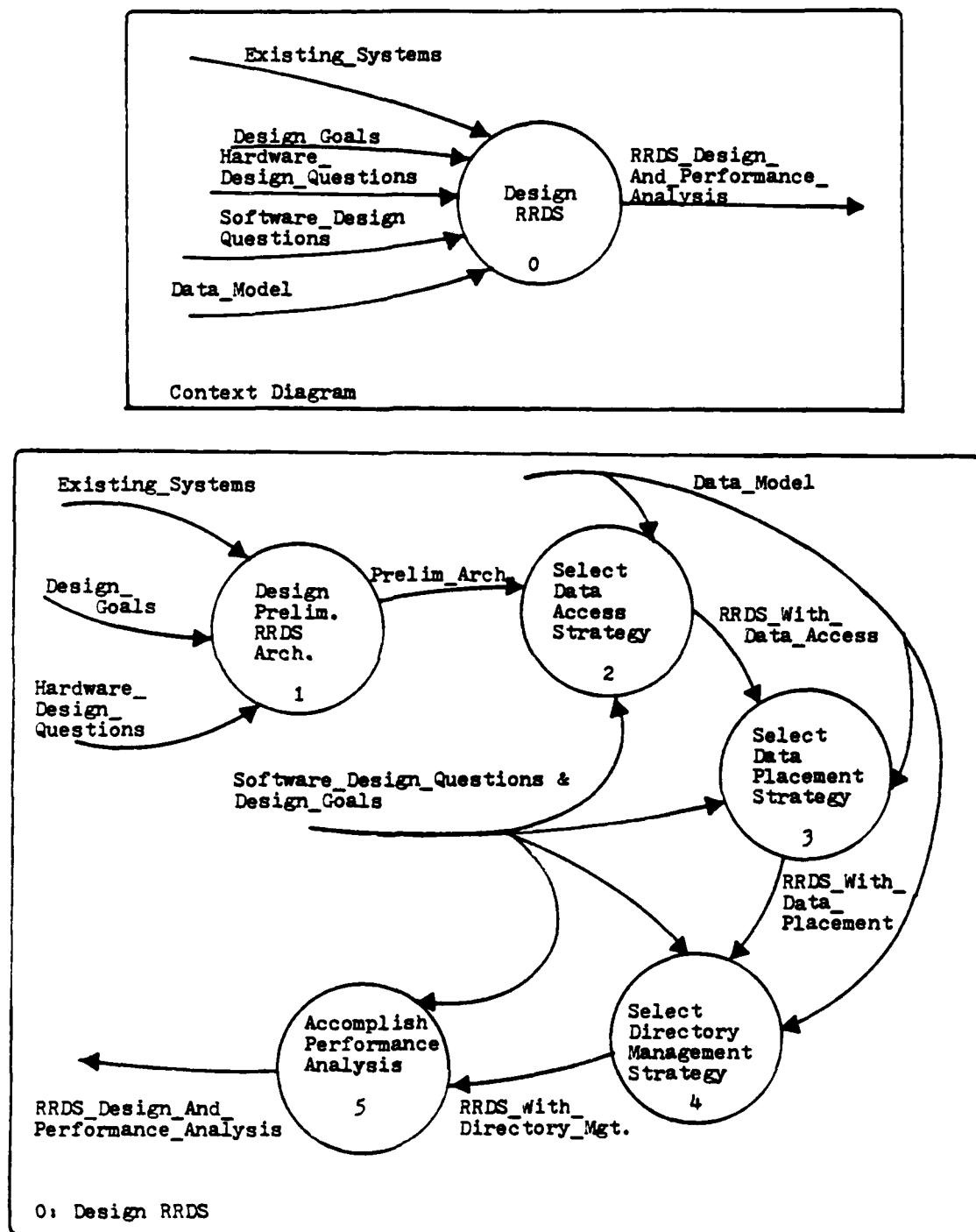


Figure 20. Five Phase RRDS Design Process

The first software issue to be resolved is the choice of a data access strategy. Many options are available including hashing, indexing, and clustering schemes. We eliminate hashing from contention due to its inefficiency in processing range queries. In designing RRDS, two approaches--clustering and indexing will be evaluated with respect to their impact on directory size and directory management overhead.

Clustering allows operations to be performed on subsets of relations instead of whole relations. These subsets are likely response sets. Each relation in the database, consists of a set of non-intersecting clusters, each cluster containing a set of records, each record having all the attributes defining the relation scheme. Clustering provides the advantages of 1) reducing the segment of a relation that needs to be processed to answer a query, and 2) providing relation subsets for distribution across the processors, enabling better use of the system's parallel processing capabilities. These advantages, however, do not come without cost. Using a clustering scheme complicates the directory management function by increasing directory complexity and size. Clustering requires directory replication as well as the maintenance of relation and cluster information. Voluminous directories could overwhelm the processors, in a very large database application, resulting in a backend limitation problem.

In the second method, called indexing, a replicated directory is not required. This allows the system directory to

be partitioned across the backend network in the same manner as the database itself. Instead of being members of relation subsets, records in an indexed organization are accessed by keys. Many different indexing schemes are available including sparse index, dense index, hierarchical organizations (e.g., tree structures), and combinations (Ullman 1982). The indexed approach has the advantage of simplicity and does not require the directory management overhead associated with defining and maintaining clusters. An indexed organization can also provide fast access, especially in a very large database environment where clusters may become large. Conversely, some indexing schemes require records to be sorted and the size of the index structure(s) can become large.

Hierarchical indexing schemes will be examined as well as the clustering approach. Tradeoffs between the two strategies will be evaluated with respect to the overall RRDS design goals, and an optimal strategy selected for implementation. The choice of a data access strategy for RRDS will also influence design decisions concerning data placement.

In order to efficiently manage a very large database an optimal data placement strategy must be integrated into the system. The sheer size of the database dictates that it be partitioned across the backend network. The goal is to employ a technique allowing all the backend processors to participate equally in the execution of each query. This approach maximizes parallelism and diminishes the specialized backend problem,

resulting in better response time. This phase of system design will examine different data placement strategies with respect to the design goals. The best placement strategy for RRDS will be chosen and integrated into the system design.

Following determination of data access and data placement strategies for RRDS, the question of directory management must be addressed. In keeping with the desire to minimize controller limitations and maximize parallel processing, the system directory should be located in and managed by the backend network. Questions which must now be answered concern how the directory will be divided among the processors and how it will be managed. Since the directory will be used in some capacity, during every query, various workload division strategies must be explored. Should the directory be fully replicated, partially replicated, or partitioned? Should a single processor be designated to perform directory management, or should the task be divided among all processors? Intuitively, it seems that the directory should be replicated and all of the backends should manage it in parallel. But, will this strategy overburden the communication network with control messages, and will the directory for a very large database be too large to replicate? These questions will be answered in phase four of the design process, and a directory management strategy proposed for RRDS.

The result of the design process, described here, will be an RRDS architecture minimizing the limitation problems characteristic of multi-backend database systems and

incorporating the complete relational data model. Integrated into the system will be data access and placement mechanisms which refine response set granularity and maximize parallel processing. Finally, a directory management strategy developed for RRDS will ensure that the overhead of directory processing does not degrade system performance.

The ultimate design goal governing the RRDS development process is performance proportional to the number of processing elements. Given a certain database we should be able to increase the number of processors and expect a proportional improvement in throughput. Also, faced with the need to increase capacity, we should be able to simply add backend processors and secondary storage and maintain the same throughput level.

The performance analysis phase of this research, presented in Chapter 8, will reveal the extent to which this design goal has been realized. The final RRDS simulation model, incorporating all the features, will be subjected to extensive experimentation for a variety of stresses and configurations. As the system environment is altered and input parameters varied, the effects upon system performance will be observed. This study will provide valuable information about performance of the system under certain conditions. For example, the throughput for a query mix which is Select-intensive may be considerably better than for one which is Insert-intensive. Information of this type will be useful in determining the best uses for such a system. This phase of the simulation effort will serve as the final

yardstick for system performance prior to actual development of the system prototype, a time when any severe limitations should already be known.

In this chapter the foundation for RRDS development was laid. A set of goals which will influence all design decisions was presented. Based upon characteristics of existing software multi-backend systems a set of hardware design questions was proposed. Rationale for choosing the relational data model, a decision with profound impact upon all aspects of RRDS design, was provided as well as a description of operations and DML to be supported. Finally, a five-phase process for producing the detailed design was presented. The methodology, which is both iterative and evolutionary, begins with a generic multi-backend system model and produces a preliminary RRDS architecture. The complete RRDS evolves from the preliminary architecture based upon software and hardware design criteria and the results of analytical and simulation models. The design process culminates in a thorough performance analysis to predict system capabilities. In the next chapter the preliminary RRDS architecture is presented.

CHAPTER 4

DEVELOPING A PRELIMINARY HARDWARE ARCHITECTURE

This chapter describes phase one of the RRDS design process. First, a generic software multi-backend database system will be developed. Different components, functions, and their relationships are identified and incorporated into the design. Each component of the generic architecture will be described relative to the six design questions outlined in Chapter 3. Next, the generic architecture will be refined, based upon the RRDS design goals, into a preliminary RRDS hardware configuration. The preliminary architecture incorporates high-level solutions for the design questions. Finally, a comparison study, based upon analytical models and simulations, is presented, comparing the performance of the preliminary RRDS architecture with three other hardware configurations. We are interested in studying the effect of the six design questions on the response time, throughput, and component utilization of the respective architectures. Once the preliminary RRDS hardware configuration is finalized, the various software design questions will be addressed in the following design phases.

A Generic Software Multi-Backend Architecture

The goal of the multi-backend approach is to offload the database management functions from the host to a network of

processors (backends). A generic multi-backend architecture is illustrated in Figure 21. System users communicate via a host processor which is not necessarily dedicated to the database management system (DBMS). The host passes queries to the controller, where actual processing begins.

The controller can range from primitive to complex and may be centralized or decentralized. It can handle a variable amount of the database management as well as the directory management. When assigning tasks to the controller, and determining its complexity, the designer must ensure that system performance is not adversely affected by the limitations of this component. Controller limitation arises when this component becomes a system bottleneck. An optimal system architecture minimizes the effects of controller limitation by reducing its workload and complexity to the lowest possible level. Ideally, the controller should handle none of the database management functions and a minimal amount of global directory management.

Two communication networks are depicted, in Figure 21, interconnecting the various system components. The complexity of these networks depends largely upon the configurations chosen for the controller, backend network, secondary memory, and the degree of connectivity between these components. Two important parameters, which must be minimized for efficient communications, are the quantity and size of messages passed across the communication networks. Communication network limitation arises when these parameters degrade system performance. Excessive

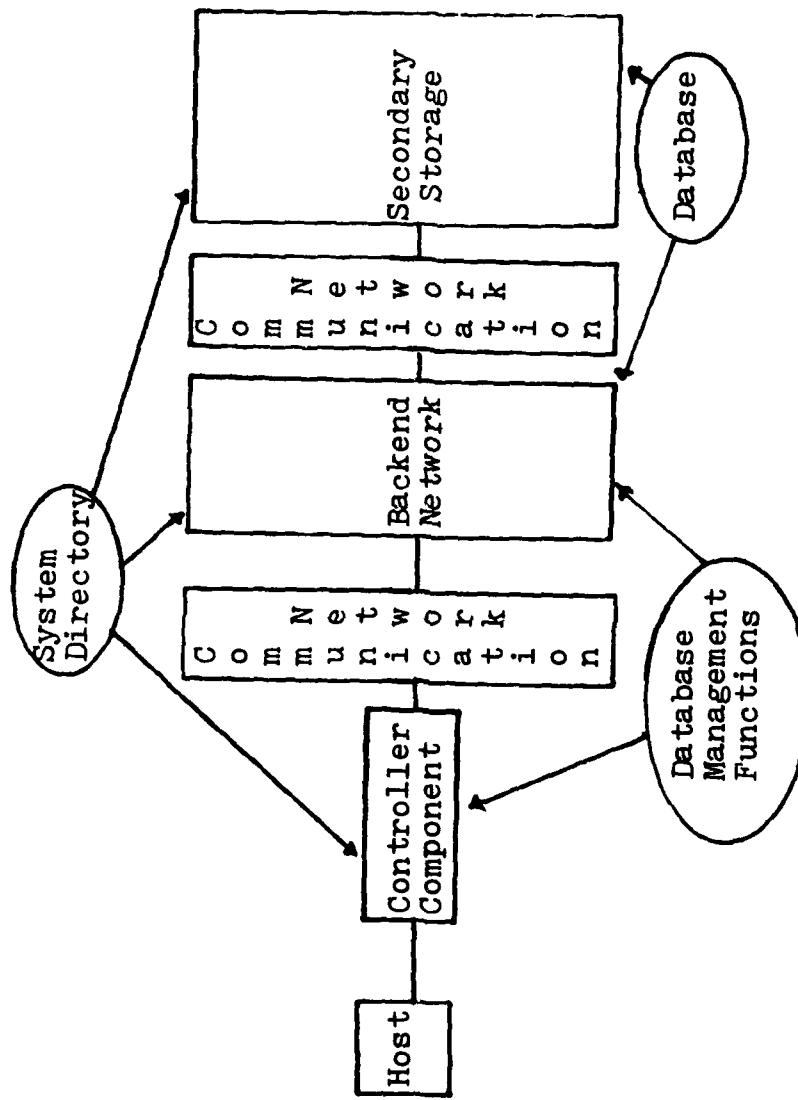


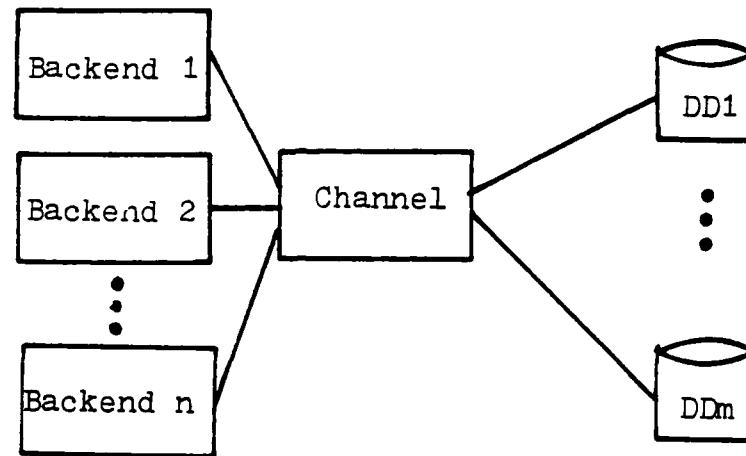
Figure 21. Generic Multi-Backend Architecture

message overhead can result in a performance anomaly where system response time actually degrades with the addition of more backends, beyond a certain number. As in all multi-processor configurations communication efficiency and speed must ensure that the benefits of parallel processing are not eclipsed by excessive control message traffic and/or messages which are too large.

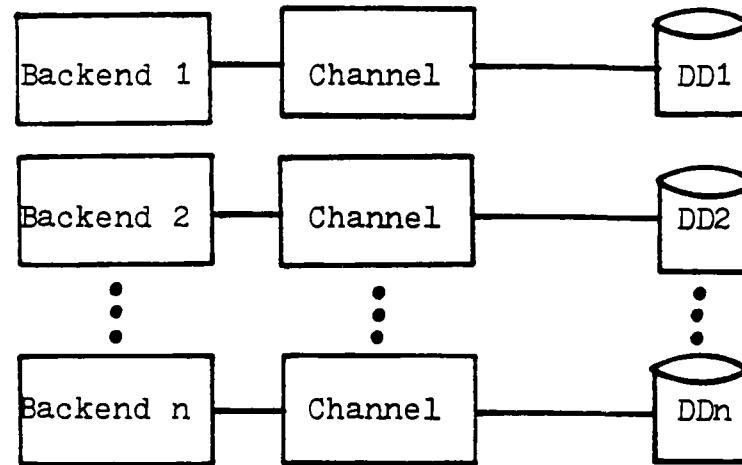
In a multi-backend system the backend processors are where the majority of the processing should be accomplished. Ideally, they should be responsible for all database management functions and most of the directory management, security, and concurrency control. Design questions which must be addressed while developing such a system include the backend specialization and backend limitation problems. Backend specialization occurs when only certain processors are dedicated to certain tasks. For example, a certain relational system architecture might designate a specific JOIN processor, or a single backend for directory management. In addition to creating a potential system bottleneck (if a majority of queries are joins, or joins on large relations), the loss of the specialized processor results in the complete loss of a certain capability (join in this case). To avoid this problem, the backends should be homogeneous, all running identical software and performing all of the DBMS functions. Backend limitation occurs when the system's performance is limited due to the number of backends participating in query processing. For instance, if a query

requires retrieval of information stored only at the first backend, only this processor is involved in processing the query, leaving all the others idle. The backend limitation problem is eliminated by maximizing parallel processing and having all the backends participate equally in processing each query.

The secondary storage capability, illustrated in Figure 21, consists of a network of channels and secondary memory devices (disk drives). As in the backend network, many connection strategies are possible ranging from having all of the disk drives dedicated to all of the backends to having single or multiple designated disk drives dedicated to specific backends. Figure 22 shows two possible configurations covering this range. This figure illustrates two problems which pertain to designing the secondary storage network--the channel limitation and the device limitation problems. Channel limitation is characteristic of systems where all of the parallel processors access all of the disk drives (i.e., no one-to-one correspondence between the backends and the disk drives). The system's throughput is limited by the speed of the access channel, because, even though there are a number of backends, they can not access different portions of the secondary memory simultaneously. In the second case, where each backend has a dedicated disk drive, the channel limitation problem has been resolved, however the system's capability is limited due to the storage capacity of the disk drives. According to Hsiao and Menon (1981a), very large databases of the magnitude of tens of billions of bytes can not



All Backends Accessing All Disk Drives (Channel Limitation)



One Dedicated Drive Per Backend (Device Limitation)

Figure 22. Channel And Device Limitation

be supported by such a configuration due to the fact that hundreds of backends would be required, making such a system cost-prohibitive. An optimal multi-backend architecture must not exhibit device limitation and channel limitation.

A Preliminary Architecture

In trying to minimize and/or eliminate the design problems discussed in the previous section, the generic architecture of Figure 21 can now be refined to produce the preliminary RRDS architecture of Figure 23.

Eliminating bottlenecks is a primary consideration in this configuration. Bottlenecks are created by the controller limitation, channel limitation, and communication network limitation problems described previously. In RRDS the controller function is accomplished by a single processor with a light workload. None of the time-consuming database management functions are carried out by the controller, and its only roles in query processing are parsing and lexical analysis of the queries, final calculation of aggregates, elimination of duplicate records, and transmission of results to the host. By reducing controller complexity and freeing it of the burdens of directory management, database manipulation, and concurrency control, the controller limitation problem has been negated.

Providing RRDS with a broadcast capability between the controller and the backends eliminates the communication network limitation problem related to the quantity of messages. In

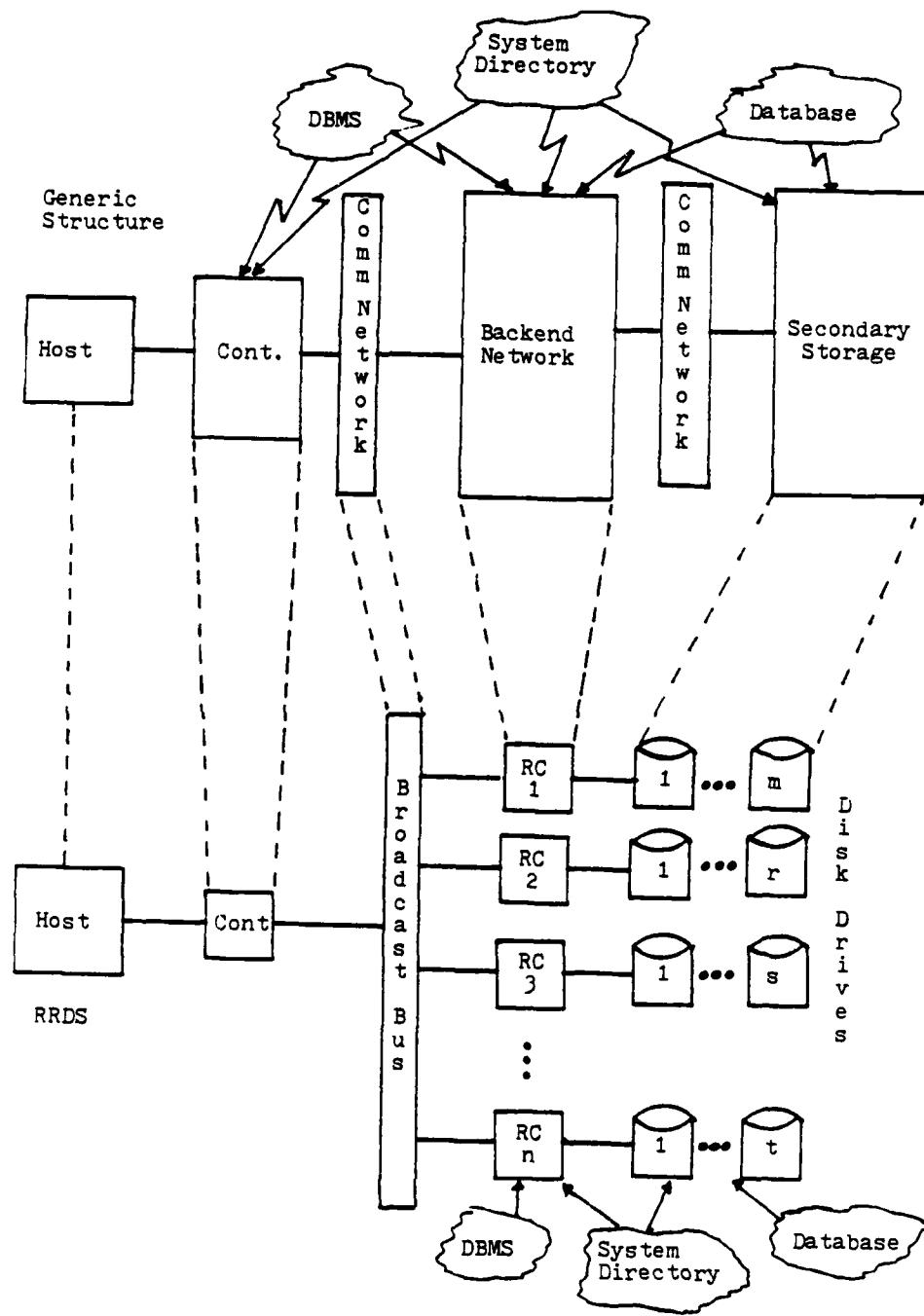


Figure 23. A Preliminary RRDS Architecture

addition, the size of messages will be limited due to the directory management strategy used for data access. In RRDS the second communication network (between the backends and secondary storage) has been virtually eliminated and consists only of direct links between each backend and its dedicated disk drives. The final potential bottleneck, channel limitation, has been eliminated by designating multiple disk drives, and one or more channels, to each backend processor.

A second design consideration addresses the problems of backend specialization, backend limitation, and device limitation. In RRDS there are no specialized backends. All of the backends, called replicated computers (RCs), execute all of the database management functions in parallel. In particular, they have exactly the same software, hence the name replicated computer. In addition to eliminating the backend specialization problem, this approach provides extensibility, eliminating the backend limitation problem. A need for more data handling capability can be met by simply adding more RCs. Device limitation is eliminated by assigning multiple disk drives to each RC. The database is partitioned evenly across all the RCs according to a data placement strategy. The system directory is also spread across the RCs according to a strategy to be determined in subsequent phases of development.

A final RRDS design goal is that the system not depend on specialized hardware. All system components are to be commercially-available, off-the-shelf hardware. This approach

will facilitate system prototyping and avoid the problem of unavailable technology exhibited by some of the systems described in Chapter 2.

Comparison of The RRDS Architecture With Alternative Approaches

In this section the proposed RRDS preliminary architecture is modeled and compared with three alternative approaches based upon characteristics of Stonebraker's Machine (Stonebraker 1978), DIRECT (DeWitt 1979), and RDBM (Auer 1980). Each model is designed to reflect certain distinct characteristics of interest for each architecture. All of the models are based upon a set of common assumptions and characteristics, for simplification. The goal is not to build exact models of the systems, but instead to create organizations similar to these machines, reflecting some of their prominent characteristics, and to observe the effects of the limitations and bottlenecks discussed in chapters 2 and 3.

The RRDS preliminary architecture is based upon a replicated computer concept with identical software running on all processors, in an MIMD mode, on a partitioned database. The three alternative architectures, to be compared with RRDS, feature an SIMD approach, an MIMD approach, and a functional-division approach. For each architecture a high-level description and block diagram is presented, providing component descriptions, component relationships, and a high-level query processing algorithm. An analytical model is developed to determine component activity times which will then be integrated into a

queueing network model implemented in the SLAM simulation language (Pritsker 1986). Following development of the architectural models a set of experiments is designed and run to examine the impact of the previously discussed design questions on system performance.

The results of this comparison will reveal if the approach taken by RRDS to solve the hardware design questions is the correct one, or if other alternatives, such as functional division or a dedicated directory backend, should be considered.

Modeling Approach, Common Assumptions, Parameters and Variables

Modeling Approach. The approach used for developing and running the alternative architecture models is to combine the capabilities and flexibility of the SLAM simulation language and the simplicity of analytical models. First, the generic multi-backend system will be modeled as an open queueing system. Next this model will be enhanced to reflect the four candidate architectures with service times determined through analytical techniques, based upon each architecture's characteristics. In order to derive meaningful results from these four diverse approaches to the multi-backend organization, certain simplifying assumptions, discussed later, were made for all the systems.

A queueing system problem may be described as a process where the arrival rate of the customers into the system, the number of servers available for these arriving customers, or both of these components are the primary subject of the study. Furthermore,

the costs associated with both the waiting time of the customers and the idle or busy times of the servers are under constant review. Queuing systems may be decomposed into three major components:

1. Number of available servers
2. Number of arriving customers
3. Behavior of servers and customers

Multi-backend database systems may be viewed as queuing systems where the queries and transactions are customers and the different system components or modules constitute servers (Hsiao 1981b; Draw 1983; Salza 1983). The specific machine's architectural characteristics will determine the manner in which the servers and customers behave.

At the highest level of abstraction the entire system can be represented as a single-queue system with one server (M/M/1). This single-queue database system is illustrated in Figure 24. Inputs to the system consist of user requests for information, or queries. The queries may be any of the relational operations and may have one or two relations as operands. In addition, they may have variable response set sizes. Other factors influencing the inputs are the query mix (e.g., 60% Select and 40% other types) and their inter-arrival times (IAT). The single server simulates the actions of the database system from the time a query is received by the controller to the time the result is sent to the host. The service time is directly affected by the system architecture and is determined by the design factors enumerated

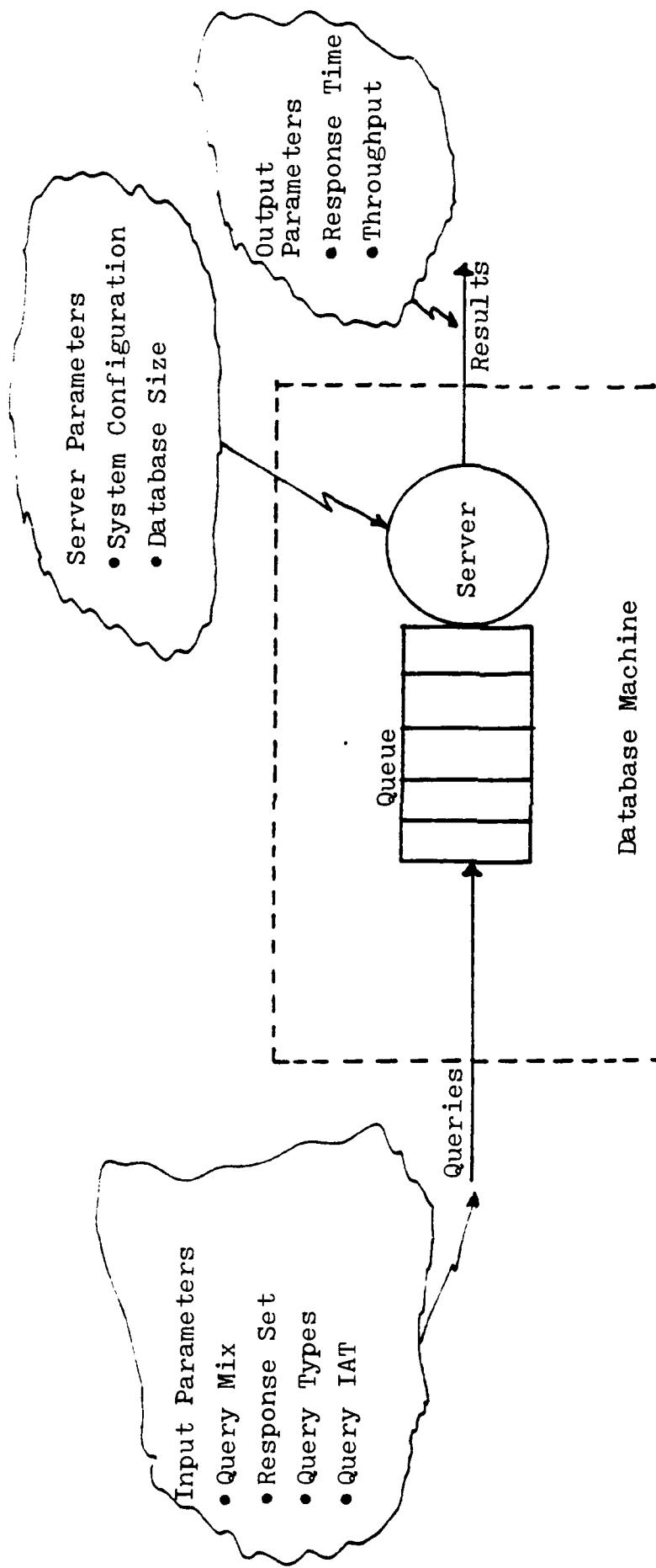


Figure 24. Single-Queue Database Machine Model

in Chapter 3. The length of the queue is determined by the interarrival rate of the queries and the service time of the database system. All of these factors influence the performance parameters such as throughput (the number of queries processed per unit time), and response time (the time required to process a query). System capacity (database size) cannot be determined at this highest level of abstraction since there is no secondary storage subsystem.

In the next iteration these concepts can be extended to the generic multi-backend database system by representing each system component as a single-queue system for processing the input data stream and a single-queue system for processing the output data stream. This model is illustrated in Figure 25. In addition to representing each system component in terms of single-queue systems processing data in each direction, some of the parameters affecting service times are illustrated. One can see that the complexity of the model increases profoundly as new details are integrated. These details, for each of the architectures to be modeled, will be provided by analytical models of each component's activity time. These activity times will be integrated, to form new models representing the different architectures, for comparison with the preliminary RRDS architecture. These models can be directly mapped into SLAM networks for simulation, results collection, and analysis.

Once the SLAM networks are developed, a workload model and a set of experiments will be designed to examine the effects of

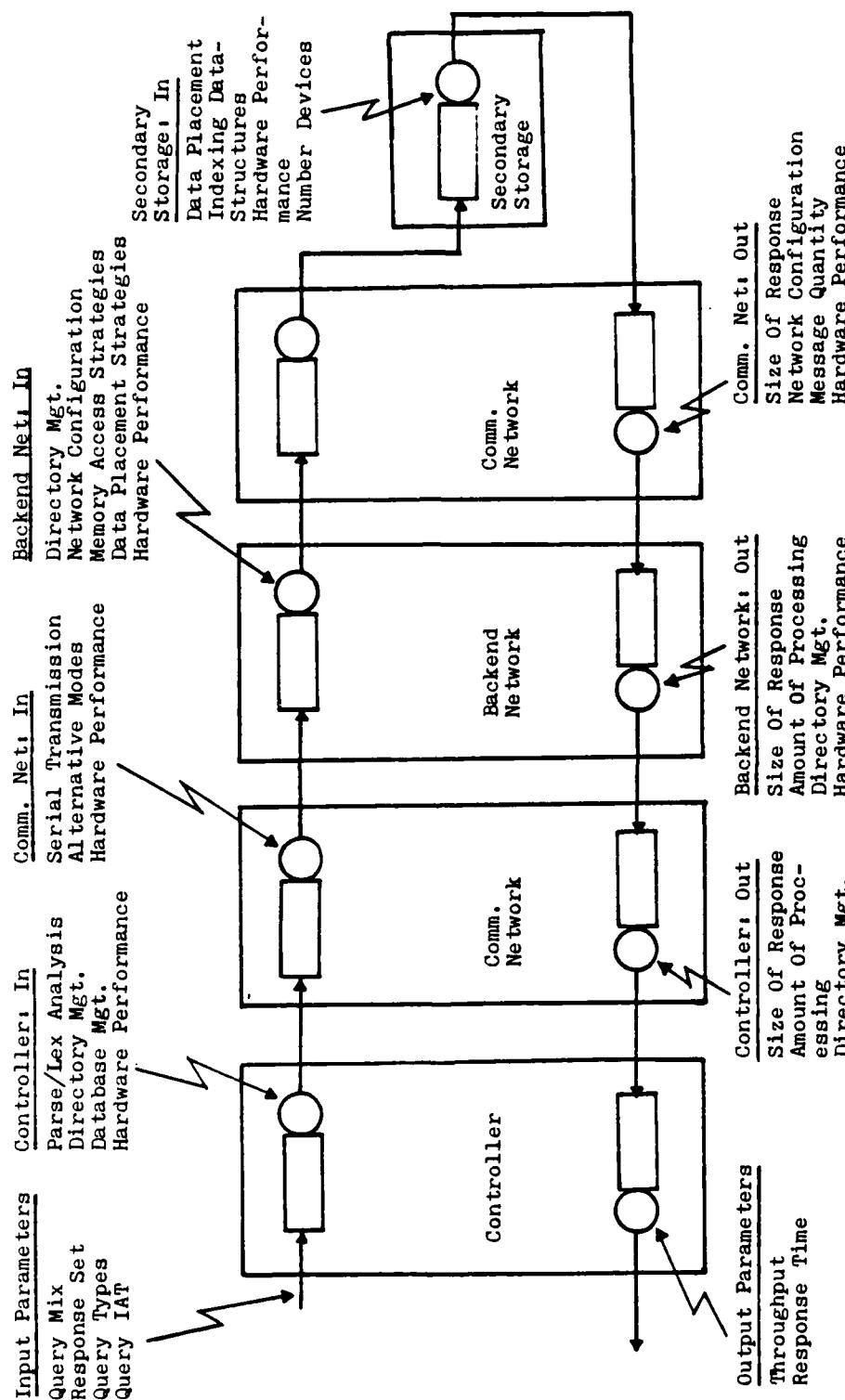


Figure 25: Generic Multi-Backend System Queuing System Model

each of the design questions from chapter 3 upon the four architecture models. Each experiment requires alteration of one or more parameters, observation of the effect on the simulation model, and collection of results for analysis.

The system models are based upon a process orientation as shown in Figure 26. This approach combines features of both the event orientation and activity scanning orientation methods (Pritsker 1986) of model construction. The process orientation should provide sufficient flexibility and capability to perform the task at hand, yet still be simple enough to facilitate development of the many models required by the effort. Languages best suited to this process-oriented approach include GPSS (Gordon 1975), SIMULA (Birtwhistle 1973), and Q-GERT (Pritsker 1977). SIMULA has been used in similar studies such as the simulation and analysis of MDBS (Hsiao 1981a, 1981b) and simulation modeling of various concurrency control mechanisms for distributed database systems (Ozsu 1983). However, SIMULA employs a statement orientation which is a subset of ALGOL. Q-GERT, on the other hand, employs nodes and branches interconnected into a network model. This is the approach used to develop our simulation models. SLAM, available here at the University of Central Florida, is a superset of Q-GERT.

In SLAM alternative modeling world views are combined to provide a unified modeling framework. The process orientation of SLAM employs a network structure (like Q-GERT) comprised of specialized symbols called nodes and branches. These symbols

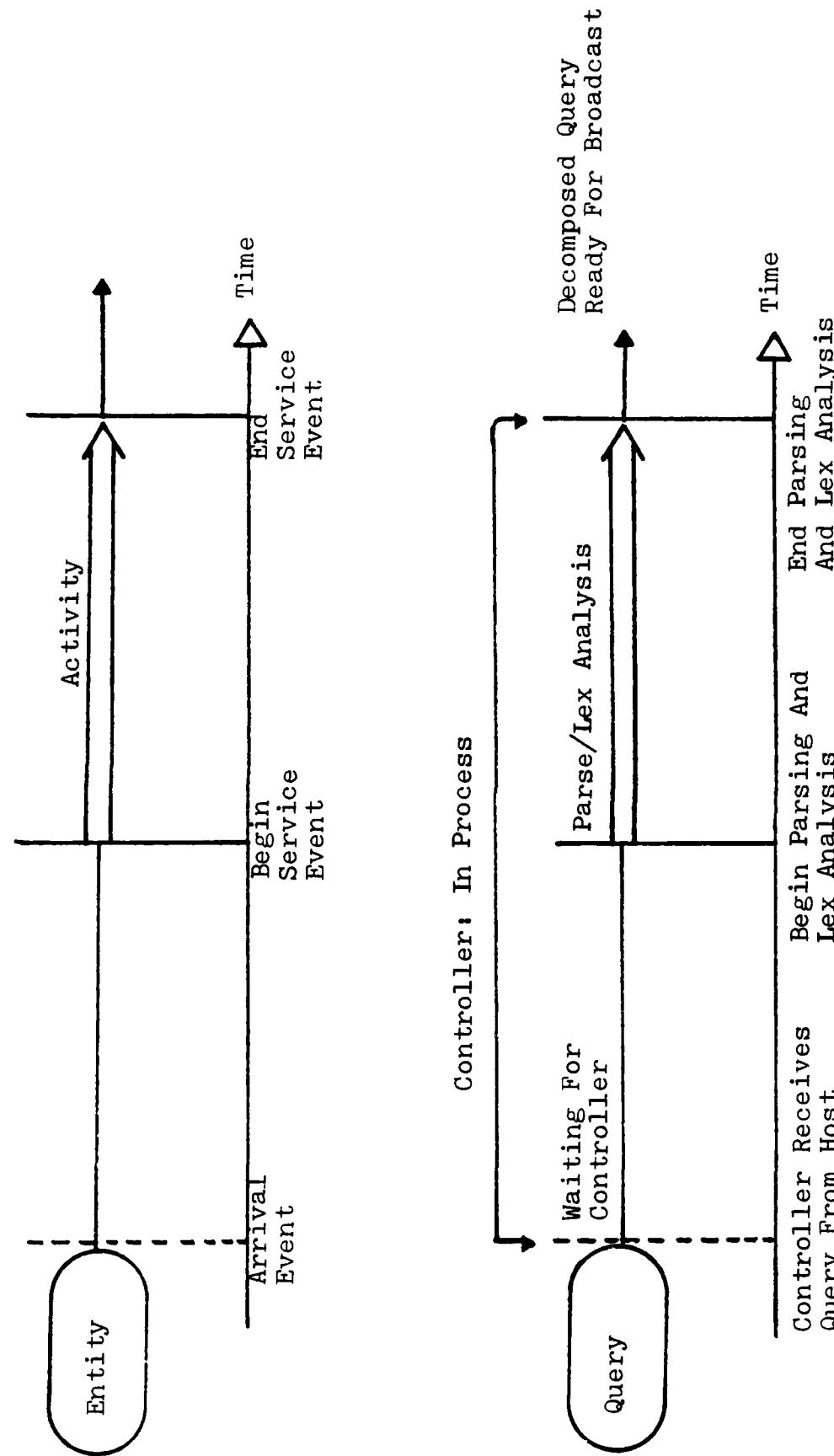


Figure 26. Relationship Between Events, Activities, Entities, And Processes

model elements in a process such as queues (e.g., queries waiting for the controller), servers (e.g., the controller), and decision points (e.g., determining which backends will process a query). The modeling task consists of combining the symbols into a network model to represent the system of interest. The entities in the system (e.g., queries and query-related items such as records) flow through the network where processes are activated and statistics collected.

The transformation of the single-queue database system context model of Figure 24 into a SLAM network is shown in Figure 27. In addition to providing outstanding network modeling capability, FORTRAN-based SLAM also allows the analyst to incorporate discrete events into the model in the form of FORTRAN subroutines. The power of these combined capabilities, along with the language's availability and report generating capabilities, make SLAM an excellent choice for this simulation effort.

Common Assumptions. In order to keep the four architecture models simple and consistent certain universal assumptions were applied. The fact that all the architectures are constructed from the same generic components allows us to observe the effect of each design approach without worrying about specific implementation details. This approach also facilitates modeling of proposed but not-implemented systems by eliminating the need to validate the models against actual operational systems. The assumptions stated here apply to all four systems. Specific details are

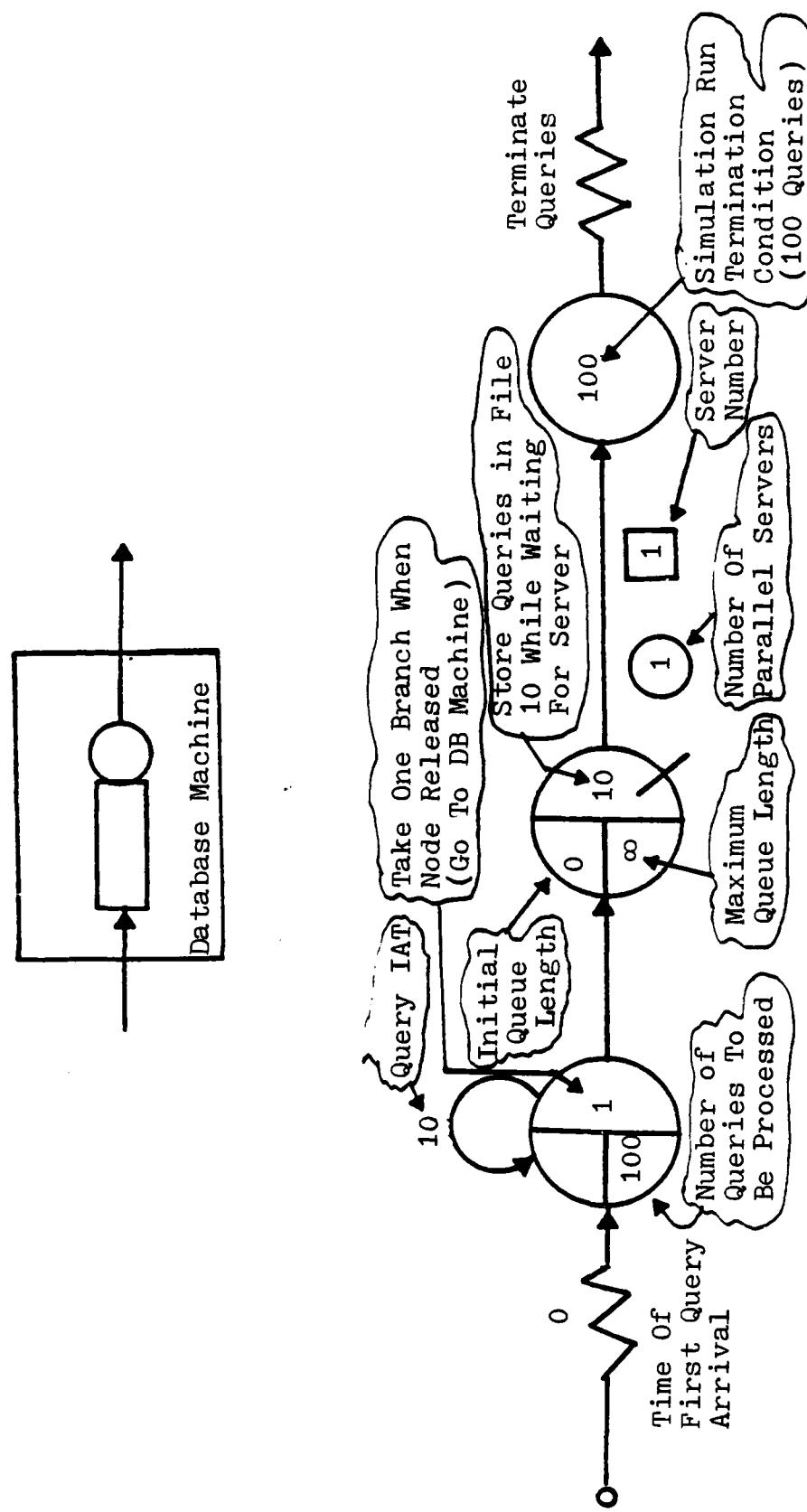


Figure 27. SLAM Network For Database Machine Context (Single Queue) Model

provided in the model descriptions for each respective architecture. The common assumptions are listed below:

1. Each system component may be modeled as a system of queues.
2. The generic architecture may be modeled as a system of queues.
3. All components are 100% reliable, therefore, reliability is not a factor in the model.
4. All components are generic (e.g., the controller in each model is the same in terms of capability and processing power).
5. Nine different relational operations are supported which can be categorized into four query types, depending on whether they take one or two relations as operands, as follows:

<u>QUERY TYPE</u>	<u>OPERATIONS</u>	<u>OPERANDS</u>
Type 1	Select/Project/Delete/Update	1
Type 2	Difference/Intersect/Join	2
Type 3	Union	2
Type 4	Insert	1

6. Critical components can be modeled as system resources.
7. All query interarrival times are accurately modeled with non-shifting exponential distributions.
8. All database records are of the same fixed length and require the same amount of CPU processing.
9. All records require the same transmission time over

communication networks.

10. All messages arrive in the order they are sent.

11. All system messages require the same transmission time.

12. All the models use the same query format and only single-predicate queries are modeled.

13. Controller and backends' primary memory will be sufficient to handle all query responses, therefore, this memory will not be modeled as a critical resource.

14. No specific data access schemes are implemented; each record requires a secondary storage block access (worst case).

15. The entire relation must be searched each time to find a specific record (worst case).

16. No concurrency control or security mechanisms are modeled.

17. In all two-relation queries, Relation R2 is the smallest (in terms of number of records).

18. Aggregate functions are not modeled.

Parameters and Variables. The following values are used for the various parameters in all the architectural models.

Tparse = 20 ms (time to parse/lexically analyze query)

Tmtrans = 8.01ms (time to transmit a message via communication network)

```
Trtrans = .01ms (time to transmit a fixed-length record)
Ta      = 25 ms (secondary storage access time)
Tr      = .053 ms (time to read a record from secondary
                   storage)
Tc      = 2.5 ms (time to access and read/write a record
                   from/to disk cache memory)
Tp      = .0426 ms (time to perform record processing in
                   backend/controller as well as time to
                   generate a system status message)
```

The following variables are used in the workload models of the various architectures. Value ranges for these variables are provided when describing the experimentation on the models.

```
|R1| : Size of relation 1 (in records) involved in query
|R2| : Size of relation 2 (in records) involved in query
|Resp1| : Size of response set from relation 1 (in records)
           satisfying query
|Resp2| : Size of response set from relation 2 (in records)
           satisfying query
n      : Number of backends in backend network
p      : Degree of parallelism
           (i.e., number of backends operating on query)
x      : Number of target relations in query
Pm    : Probability that records of target relation
           are in primary memory
IAT   : Query interarrival time (in milliseconds)
```

An SIMD Architecture Model

The first architecture studied is based upon Stonebraker's Machine (Hsiao 1981a). This machine was designed for SIMD

processing and incorporated a "special backend" for directory management. It consists of a central controller and multiple backends, each with one dedicated disk drive. The backends and controller communicate via a non-broadcast bus. A query may be processed by one or more backends and backend processors do not support concurrent request execution. In addition, this system supports only queries accessing a single relation (Hsiao 1981a). As described in Chapter 2, this system exhibits backend, controller and device limitation problems.

Our model, called S-Arch (SIMD Architecture), is illustrated in Figure 28. Based upon Stonebraker's Machine, it features a single controller and multiple backend processors, communicating via a non-broadcast bus. The system directory is located entirely at backend-1, the special backend. For simplicity, in S-Arch the directory consists only of a table specifying which backend contains each relation of the database. Each relation is stored entirely at a single backend. Parallelism is achieved when two, or more, queries are operating on different relations at different backends simultaneously.

Query processing in S-Arch is illustrated in Figure 28 which depicts the actions taken from the receipt of a query by the controller, to the point where results are passed back to the host. In the graph of Figure 28 each node represents a system component and the arcs represent the time needed for the component to perform the desired service. The designation used for each node is a large "Q" denoting the fact that these are

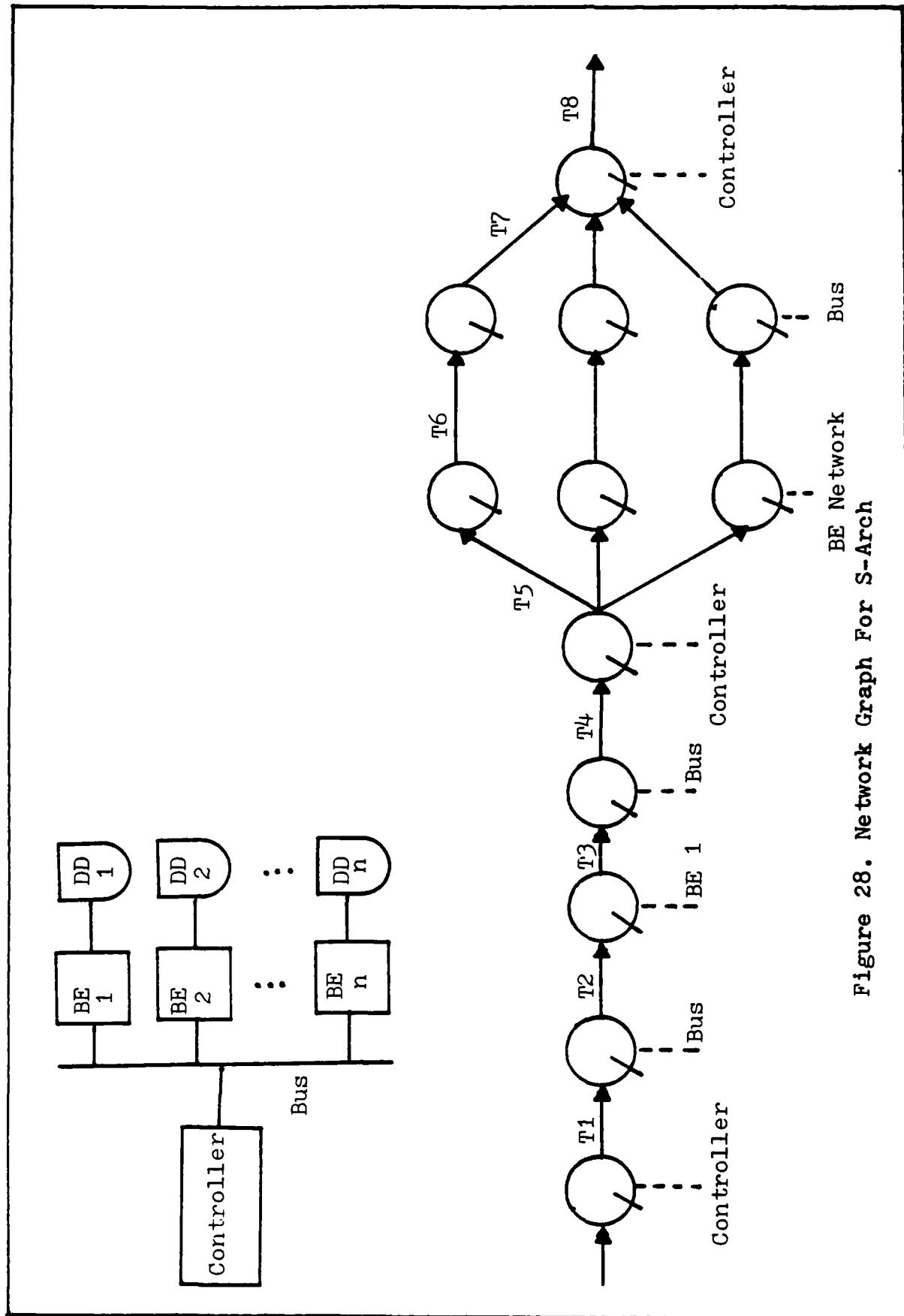


Figure 28. Network Graph For S-Arch

potential bottleneck points in the system, where queues of requests may form. Queries arrive at the controller processor where they are parsed and passed to the special backend (backend-1) for directory processing. Backend-1 consults the directory to determine which backend contains the target relation of the query and passes this information back to the controller. The controller then transmits the query to the appropriate backend for processing. Backend query processing consists of retrieving the relation from disk, selecting the appropriate records and performing the desired operation on them. In the event of an insert operation, the location for insertion is determined and the record inserted. After the backend has performed the desired operation on the records of the response set, the results are returned, via the bus, to the controller where they are collected into a temporary relation and forwarded to the user at the host.

We can now derive an expression for S-Arch query processing time (SQP).

$$SQP = \sum_{i=1}^8 T_i$$

where:

T_1 = Time to parse query.

= T_{parse}

T_2 = Time to send query to special backend with directory.

= T_{mtrans}

T3 = Time to access the directory located on backend-1 secondary storage.

= nTa

T4 = Time to send message with backend required to process the query to the controller.

= $Tmtrans$

T5 = Time to send query to appropriate backend for processing.

= $Tmtrans$.

T6 = For Type-1 queries:

Time to search relation R1 for response set records and to read response set records from the disk drive into the processor buffer plus time to perform desired operation on records of response set.

= $Ta(|R1|) + Tr(|Resp1|) + Tp(|Resp1|)$

= $Ta(|R1|) + |Resp1|(Tr + Tp)$

For Type-4 queries:

Time to search relation R1 for position to insert new record and perform insertion plus time to generate "insertion complete" status message.

= $Ta(|R1|) + Tr + Tp$

T7 = For Type-1 queries:

Time to send results of query back to controller.

= $Trtrans(|Resp1|)$

For Type-4 queries:

Time to send "insertion complete" status message to controller.

= $Tmtrans$

T8 = For Type-1 queries:

Time required by controller to receive and collect results into temporary relation for transmission to host.

= $T_p(|Resp1|)$

For Type-4 queries:

Time to generate "insertion complete" status message for transmission to host.

= T_p

So, for Type-1 queries:

$$\begin{aligned} SQP1 &= T_{parse} + T_{mtrans} + nT_a + T_{mtrans} + T_{mtrans} + T_a(|R1|) + \\ &\quad |Resp1|(T_r + T_p) + T_{rtrans}(|Resp1|) + T_p(|Resp1|) \\ &= T_{parse} + 3T_{mtrans} + T_a(n + |R1|) + \\ &\quad (T_r + 2T_p + T_{rtrans})(|Resp1|) \end{aligned}$$

And, for Type-4 queries:

$$\begin{aligned} SQP4 &= T_{parse} + T_{mtrans} + nT_a + T_{mtrans} + T_{mtrans} + T_a(|R1|) + \\ &\quad T_r + T_p + T_{mtrans} + T_p \\ &= T_{parse} + 4T_{mtrans} + T_a(n + |R1|) + T_r + 2T_p \end{aligned}$$

In order to analyze the performance of S-Arch under a variety of query loads an open queuing network simulation model was developed. The SLAM model, presented in Appendix C, facilitates modeling of all the system activities described as well as accounting for queuing delays at each point in the system. In addition the degree of inter-query parallelism can be varied to observe the effects upon response time. We are specifically interested in observing the effects of the

specialized backend, controller and backend limitations, and the non-broadcast bus on system response time, throughput, and component utilization.

The simulation model evolved directly from the network of Figure 28. Components of S-Arch are represented, in the simulation model, as resources and include the controller, bus, and backend processors. Type-1 and type-4 queries are generated according to the workload model described in the experimentation section of this chapter (Section 4.3.6). In addition to generating queries of both types their interarrival times (IAT) may be varied as well as the quantities of each type query. These capabilities provide variation of the system load as well as the query mix to observe effects on performance.

Each query carries four attributes denoting mark time, query type, response set cardinality, and target relation cardinality. Query service, by each component, consists of three phases. First, the query enters a queue where it waits for the desired resource (i.e., controller, bus, or a specific backend) to become available. This queue is modeled as an AWAIT node for which statistics are accumulated, by SLAM, providing information about bottlenecks and resource utilization. Second, when the resource is available it is allocated and the query is serviced for some activity time (i.e., times T1 - T8). The type attribute associated with each query is used to determine which activity branch is taken. Activity time is a function of the system parameters and other query attributes. Finally, when service is

complete the resource is relinquished, for the next query, by a FREE node. The simulation model consists of a network of these structures arranged according to the network of Figure 28. Activities which require no time are used to connect the network and direct queries to appropriate components. Inter-query parallelism can be varied by altering the probability values of the activity branches emanating from the bus structure to the backend network structures. This capability allows us to observe the benefits of the SIMD approach of this architecture.

The performance of the S-Arch model will be compared with the performance of three other models, including the preliminary RRDS architecture model, in the results section of this chapter. All of the simulation models will be subjected to a set of experiments, designed to illustrate the effects of the six design questions discussed in Chapter 3. The experiments are described, in detail, in sections 4.3.6 and 4.3.7.

An MIMD Architecture Model

The second architecture modeled, based upon DIRECT (DeWitt 1979), features an MIMD approach supporting both inter and intra-query parallelism. All of the relational operations are supported by this system. DIRECT consists of a number of processing units (PUs) responsible for performing DBMS functions on data read into a multi-port-memory (MPM) from disk. The central controller is responsible for parsing queries, directory management, and assigning queries to PUs. Unlike Stonebraker's

Machine, which utilizes a serial bus, DIRECT is built around a broadcast capability. Despite this fact control message traffic is high as well as traffic due to moving records from the disk to the MPM for PU access. As noted in Chapter 2, this system suffers from controller limitation and message traffic overhead.

Our model, called M-Arch (MIMD Architecture), is illustrated in Figure 29. It features a single controller and multiple processing units, communicating via a broadcast bus. The system directory is located in, and managed by, the controller. The database is stored in the disk and as relations are required for processing they are moved via the Mlink to the MPM. For the purpose of simplicity we assume the MPM has a port for each of the PUs. In addition, PUs can read/write data from/to the MPM simultaneously via the ports. It is also assumed that the MPM has the capacity to hold the target relations in their entirety.

Query processing in M-Arch is also illustrated in Figure 29. As queries are received by the controller they are parsed and the system directory is consulted to determine the addresses of the target relations. Once the target relations are located, the controller determines which PU(s) will perform the operation on the records and schedules the appropriate PU(s). The controller can vary the degree of parallelism on each query by assigning different numbers of PUs.

If the target relation is not already in the MPM the controller must notify the disk device, via the bus, that the

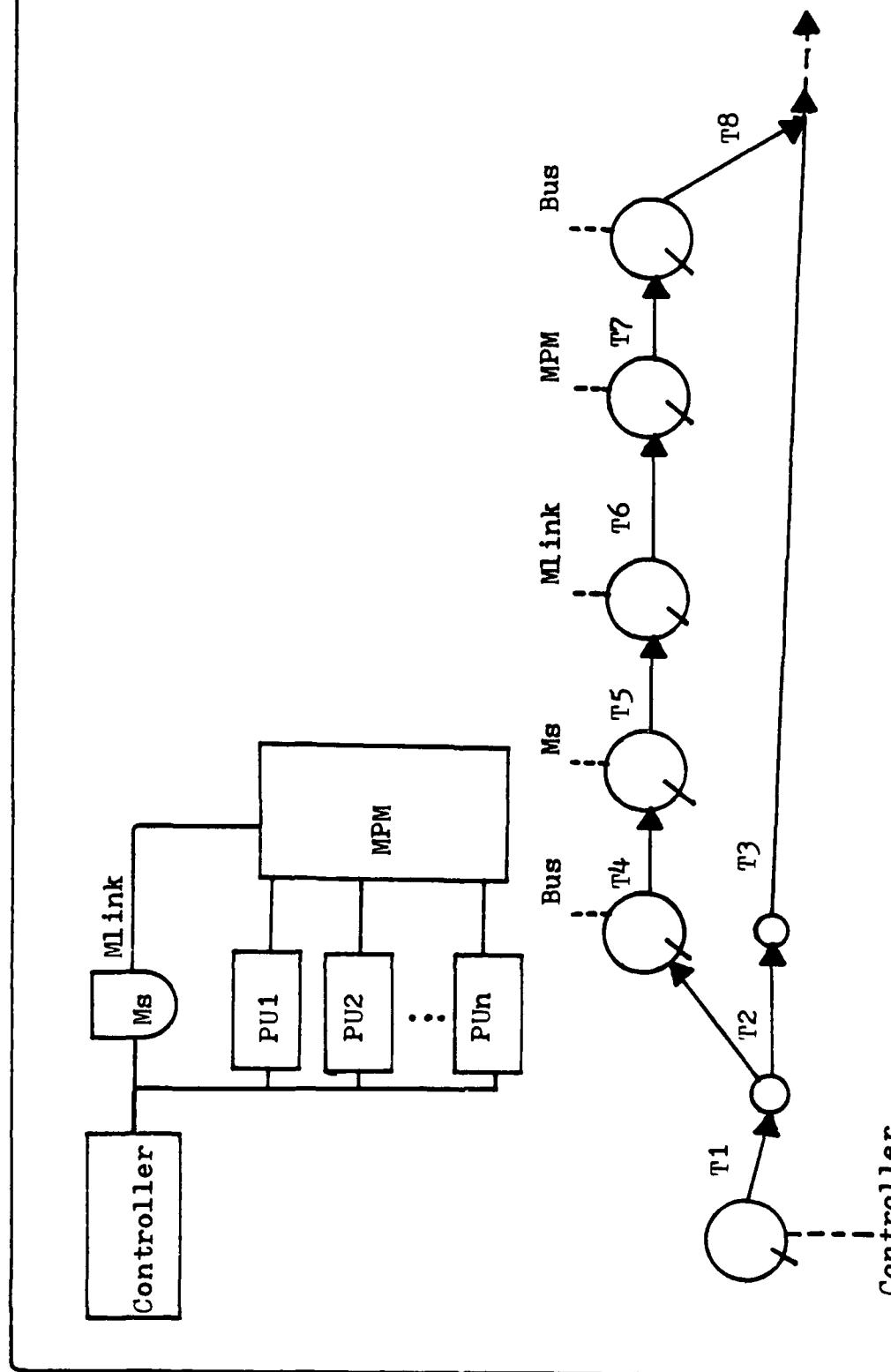


Figure 29. Network Graph For M-Arch

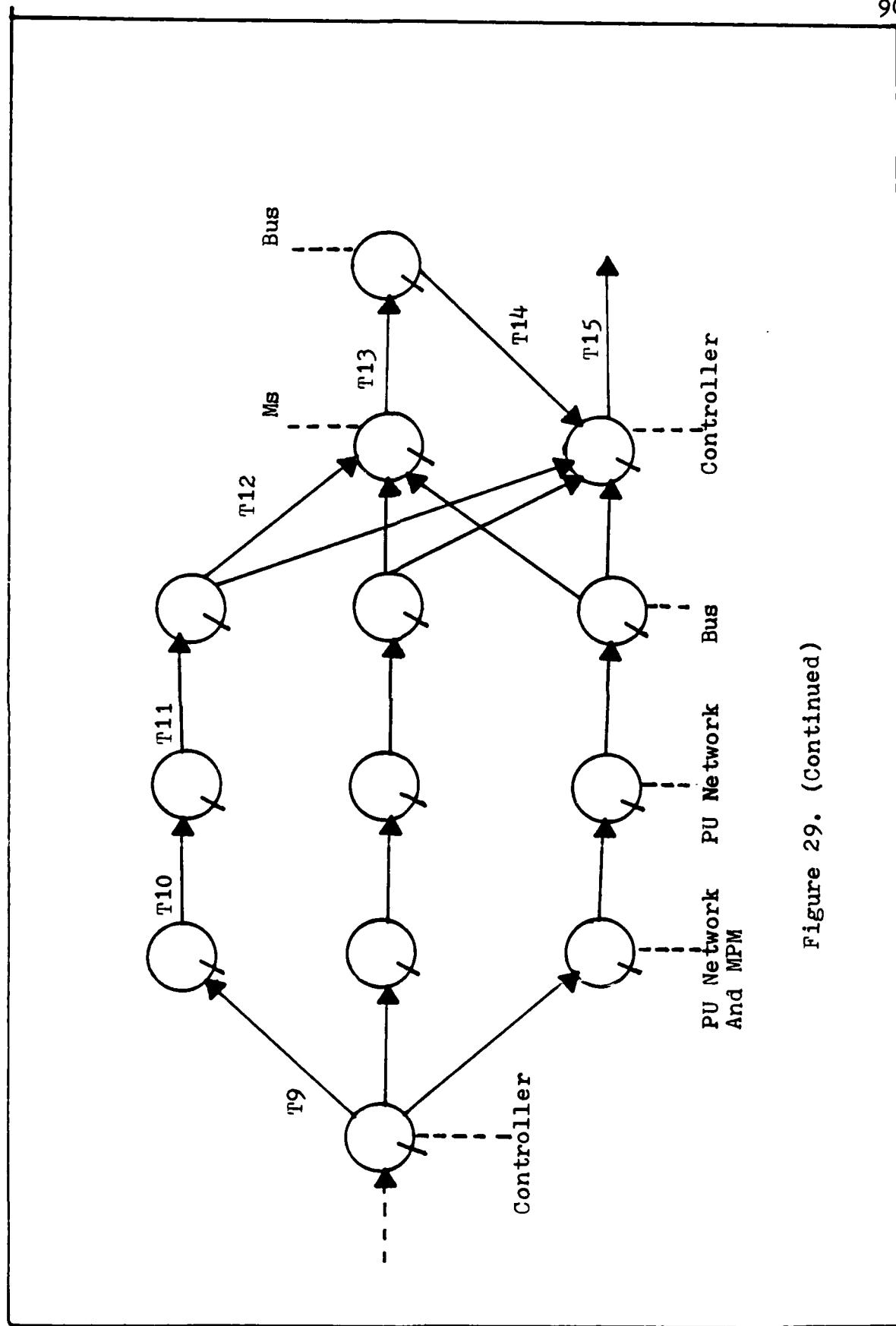


Figure 29. (Continued)

records of the relation must be moved to the MPM for access by the PUs. Records are read from the disk into the MPM via the comm link. (It is assumed that the time to perform this record transfer will always dominate the time required by the controller to select and schedule PUs.) When this process is complete the controller is notified that the query can be dispatched to the PUs for processing. Upon receipt of the query, the assigned PUs read the records of the relations, in parallel, via the MPM ports and perform the required operation. Once the processing of modification queries (Insert, Update, and Delete) is complete results are dispatched to disk via the comm link to change the database. Once the update of the database on disk is complete the "operation complete" status message is generated and returned to the controller. When processing of retrieve queries is finished the results are forwarded directly to the controller where they are collected into a temporary relation and sent to the host.

We can now derive an expression for M-Arch query processing time (MQP).

$$MQP = T1 + [Pm(T2+T3)] + (1 - Pm)[\sum_{i=4}^8 Ti] + [\sum_{i=9}^{12} Ti] + \\ Piud(T13+T14) + T15$$

Where:

Piud = probability that query is an insert, update, or delete.

T1 = Time to parse query and determine location of target relations.

= Tparse + xTp

T2 = Time to determine which PUs will participate in query processing.

= nTp

T3 = Time to schedule PUs.

= pTp

T4 = Time to notify the secondary memory that a relation must be transferred to the MPM.

= Tmtrans

T5 = Time to access and read records from secondary memory.

For Type-1 and Type-4 queries:

= |R1|Ta + |R1|Tr

For Type-2 and Type-3 queries:

= (|R1|+|R2|)Ta + (|R1|+|R2|)Tr

T6 = Time to transfer records across MLink to the MPM.

For Type-1 and Type-4 queries

= |R1|Trtrans

For Type-2 and Type-3 queries:

= (|R1|+|R2|)Trtrans

T7 = Time to read records into the MPM.

For Type-1 and Type-4 queries:

= $|R1|Tc$

For Type-2 and Type-3 queries:

= $(|R1| + |R2|)Tc$

$T8$ = Time to notify controller that the MPM contains the target relations.

= $Tmtrans$

$T9$ = Time to broadcast query to the PU network.

= $Tmtrans$

$T10$ = Time to read the records of the target relations into the PUs for processing.

For Type-1 and Type-4 queries:

= $(|R1|/p)Tc$

For Type-2 and Type-3 queries:

= $[(|R1| + |R2|)/p]Tc$

$T11$ = Time for PUs to perform query processing on relations, consisting of time to search relation fragments for the records of the response set and the time to perform the operation on the response set records.

For Type-1 queries:

= $(|R1|/p)Tp + (|Resp1|/p)Tp$

For Type-2 queries (Join):

= $[(|R1| + |R2|)/p]Tp + [(|Resp1| * |Resp2|)/p]Tp$

For Type-2 queries (Difference/Intersect):

= $[(|R1| + |R2|)/p]Tp + (|Resp1|/p)Tp$

For Type-3 queries:

= $[(|R1| + |R2|)/p]T_p$

For Type-4 queries:

= $(|R1|/p)T_p$

T12 = For Type-1 queries (Update/Delete) and Type-4 queries, time to send updated results to the secondary memory via the comm link.

= $|Resp1|Trtrans$

For Type-1 queries (Select/Project), Type-2, and Type-3 queries, time to send results to controller to be collected into a temporary relation and forwarded to the host.

For Type-1 queries:

= $|Resp1|Trtrans$

For Type-2 queries (Join):

= $(|Resp1| * |Resp2|)Trtrans$

For Type-2 queries (Difference/Intersect):

= $|Resp1|Trtrans$

For Type-3 queries:

= $(|R1| + |R2|)Trtrans$

T13 - T14 (Apply to Type-1 update and delete and Type-4 queries).

T13 = Time to update the database in secondary memory.

= $|Resp1|Ta + |Resp1|Tr$

T14 = Time to generate and transmit "operation complete"

```
status message to controller.  
= Tmtrans  
  
T15 = Time to assemble results (response) into temporary  
relation for host.  
  
For Type-1 queries:  
= |Resp1|Tp  
  
For Type-2 queries (Join):  
= (|Resp1| * |Resp2|)Tp  
  
For Type-2 queries (Intersect/Difference):  
= |Resp1|Tp  
  
For Type-3 queries:  
= (|R1| + |R2|)Tp  
  
For Type-4 queries:  
= 0
```

So, the query processing time (MQP) is as follows:

For Type-1 queries:

Select and Project:

```
MQP1a = Tparse + xTp + Pm[Tp(n+p)] +  
(1-Pm)[2Tmtrans + |R1|(Ta+Tr+Trtrans+Tc)] +  
Tmtrans + Tp(|R1|/p + |Resp1|/p + |Resp1|) +  
Tc(|R1|/p) + Trtrans(|Resp1|)
```

Update and Delete:

```
MQP1b = Tparse + xTp + Pm(Tp)(n+p) +
```

$$(1-Pm) [2Tmtrans + |R1|(Ta+Tr+Trtrans+Tc) + 2Tmtrans + Tp(|R1|/p + |Resp1|/p + |Resp1|) + Tc(|R1|/p) + |Resp1|(Trtrans + Ta + Tr)]$$

For Type-2 queries:

Join:

$$\begin{aligned} MQP2a = & Tparse + xTp + Pm(Tp)(n+p) + \\ & (1-Pm) [2Tmtrans + (|R1|+|R2|)(Ta+Tr+Trtrans+Tc)] + \\ & Tmtrans + Tc[(|R1|+|R2|)/p] + Tp[|R1|/p + |R2| + \\ & (|Resp1|*|Resp2|)/p + (|Resp1|*|Resp2|)] + \\ & Trtrans[(|Resp1|*|Resp2|)] \end{aligned}$$

Intersect and Difference:

$$\begin{aligned} MQP2b = & Tparse + xTp + Pm(Tp)(n+p) + \\ & (1-Pm) [2Tmtrans + (|R1|+|R2|)(Ta+Tr+Trtrans+Tc)] + \\ & Tmtrans + Tc[(|R1|+|R2|)/p] + Tp[(|R1|+|R2|)/p + \\ & |Resp1|/p + |Resp1|] + \\ & Trtrans(|Resp1|) \end{aligned}$$

For Type-3 queries:

$$\begin{aligned} MQP3 = & Tparse + xTp + Pm(Tp)(n+p) + \\ & (1-Pm) [2Tmtrans + (|R1|+|R2|)(Ta+Tr+Trtrans+Tc)] + \\ & Tmtrans + [(|R1|+|R2|)/p](Tc+Tp) + \\ & (|R1|+|R2|)(Trtrans+Tp) \end{aligned}$$

For Type-4 queries:

$$\begin{aligned} MQP4 = & Tparse + xTp + Pm(Tp)(n+p) + \\ & (1-Pm) [2Tmtrans + |R1|(Ta+Tr+Trtrans+Tc)] + 2Tmtrans + \end{aligned}$$

$$(|R1|/p)(Tc+Tp) + (|Resp1|)(Trtrans+Ta+Tr)$$

The complete M-Arch simulation model is presented in Appendix C. As before, system components are modeled as resources and include the controller, bus, PUs, secondary memory, MPM and ports, and the MLink. In this model queries of all four types are generated according to the workload model. Each query carries attributes denoting mark time, query type, query identification, response set cardinalities, target relation cardinalities, and the degree of parallelism. As in the S-Arch model, query service by each component consists of three phases. Intra-query parallelism is a function of the degree of parallelism attribute. If a query does not require all the PUs for execution, the remaining PUs may be assigned by the controller to other queries, facilitating inter-query parallelism. The M-Arch simulation model was subjected to the same set of experiments as the other three models and the results are described in Section 4.6.7.

A Functional Specialization Architecture Model

A different approach for database system design is one based upon functional division. The third architectural configuration modeled is based upon the functional specialization of RDBM (Auer 1980). In RDBM two-relation operations such as Join are performed by a separate processor called interrecord processor (IRP). One-relation queries are performed, in parallel, by a set

of processors called restriction and update processors (RUPs), which read data in a page buffer receiving input from secondary and primary memory components. The central controller in RDBM supervises execution of all queries from start to finish as well as performing parsing and query analysis. In Chapter 2 it was noted that RDBM suffers from controller limitation, software specialization, and channel limitation.

Our model, called F-Arch (Functional Division Architecture), is illustrated in Figure 30. A simplified version of RDBM, this model features a central controller which parses and analyzes queries and supervises their execution. All two-relation queries are handled by the IRP which reads relation records from the primary memory (Mp). One-relation queries are processed, in parallel, by the RUPs which read records from the buffer via MLink-1. The RUPs return their results to the controller via the RUP-Controller Bus (RCBus) and the I-Bus. The database is stored in the disk drives of the secondary memory (Ms) module and loaded into the buffer via MLink-2 and into Mp via the I-Bus. Unlike the previously discussed models, in F-Arch all the RUPs operate on all the one-relation queries. In addition, they are all capable of reading records from the buffer in parallel.

Query processing in F-Arch is also illustrated in Figure 30. As queries are received by the controller they are parsed and analyzed. The controller is responsible for determining whether to route the query to the IRP or the RUPs and for initiating the transfer of records from the Ms to the buffer and/or Mp. Once

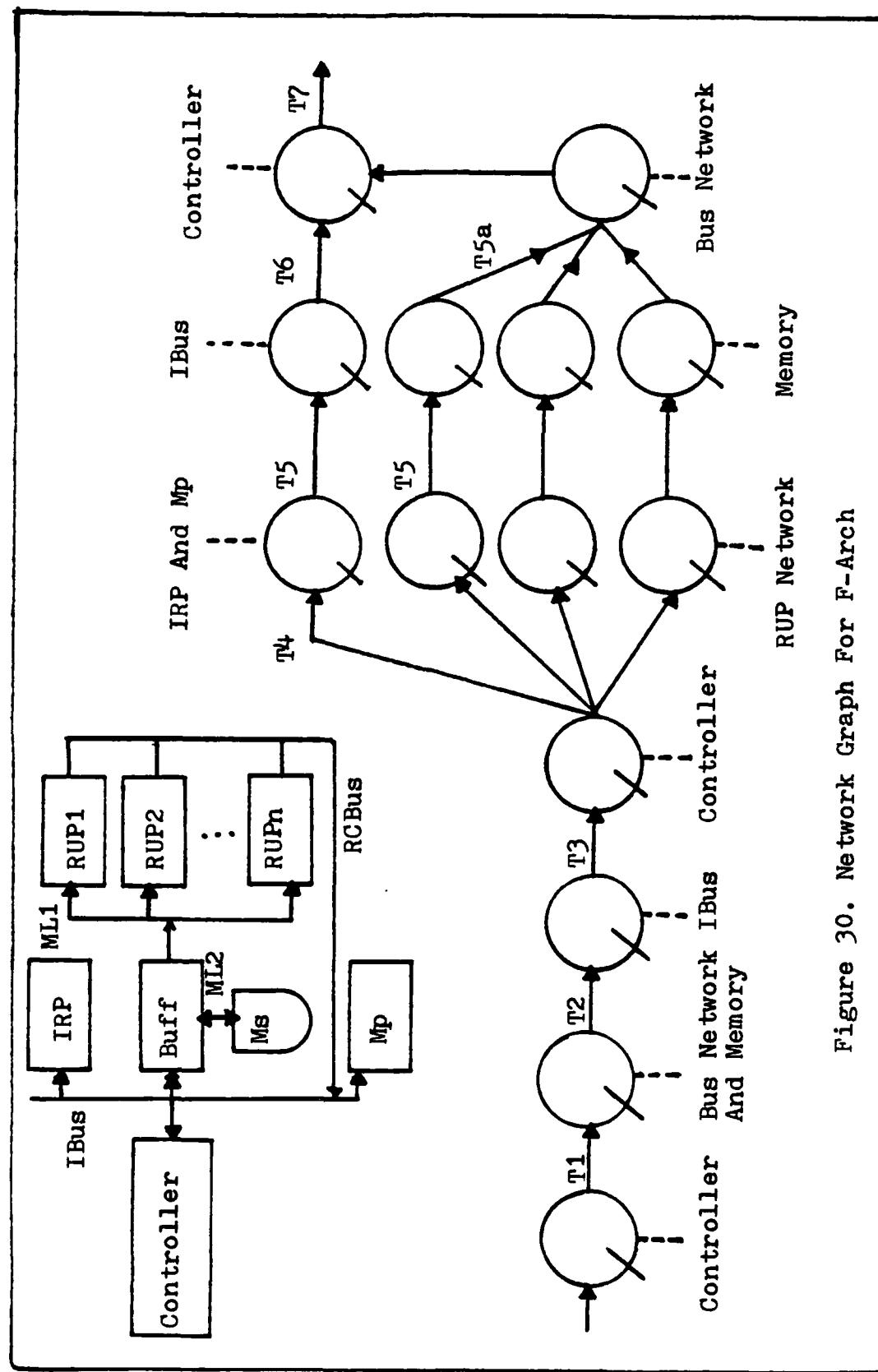


Figure 30. Network Graph For F-Arch

the relations are loaded from Ms, the controller routes the query to the appropriate processors for execution via the bus network. Two-relation queries are then processed by the IRP and results forwarded to the controller where they are collected into a temporary relation. One-relation query processing is performed in parallel by the RUPs and the results are returned to the controller to be collected into a temporary relation for the user. If the query requires a change to the database (Insert, Delete, Update) the appropriate data in Ms is modified and an "operation complete" status message is returned to the controller.

We can now derive an expression for F-Arch query processing time (FQP).

$$FQP = T1 + (1-Pm) [T2 + T3] + \sum_{i=1}^7 T_i + Piud(T5a)$$

Where:

Piud = probability that query is an insert, update or delete.

T1 = Time to parse query, set up appropriate processors and memory components and prepare processor activation messages.

For Type-1 and Type-4 queries:

$$= Tparse + (n+3)Tp + (n+2)Tp$$

For Type-2 and Type-3 queries:

$$= Tparse + (n+3)Tp + 3Tp$$

T2 = Time to load target relation(s) into buffer or Mp depending upon query type. The Ms is notified that a relation needs to be transferred, then it is

accessed and records read into the buffer. If transfer into Mp is required the relation(s) are then transferred from the buffer into Mp.

For Type-1 and Type-4 queries:

= $T_{mtrans} + |R1|(T_a + T_r + T_{rtrans})$

For Type-2 and Type-3 queries:

= $2T_{mtrans} + (|R1| + |R2|)(T_a + T_r + 2T_{rtrans} + T_c)$

T_3 = Time to notify controller that appropriate memory elements are loaded.

= T_{mtrans}

T_4 = Time to send query to the appropriate processors for execution.

For Type-1 and Type-4 queries:

= $n(T_{mtrans})$

For Type-2 and Type-3 queries:

= T_{mtrans}

T_5 = Time to perform query processing in either the IRP or the RUP network.

For Type-1 and Type-4 queries, time to read records from the buffer across MLink-1 and perform the designated operation by the RUPs, in parallel:

= $(|R1|/n)T_{rtrans} + (|Resp1|/n)T_p$

For Type-2 and Type-3, time to read records from Mp and perform the designated operation by the IRP:

For Type-2 queries (Join):

= $(|R1| + |R2|)T_{rtrans} + (|Resp1| * |Resp2|)T_p$

For Type-2 queries (Intersect and Difference):

= $(|R1| + |R2|)T_{rtrans} + (|Resp1|)T_p$

For Type-3 queries:

= $(|R1| + |R2|)(Trtrans + Tp)$

T6 = Time to send results of query to controller.

For Type-1 queries:

= $(|Resp1|/n)Trtrans$

For Type-2 queries (Join):

= $(|Resp1| * |Resp2|)Trtrans$

For Type-2 queries (Intersect and Difference):

= $(|Resp1|)Trtrans$

For Type-3 queries:

= $(|R1| + |R2|)Trtrans$

For Type-4 queries:

= 0

T7 = Time to collect results into a temporary relation to be sent to the host.

For Type-1 queries:

= $(|Resp1|)Tp$

For Type-2 queries (Join):

= $(|Resp1| * |Resp2|)Tp$

For Type-2 queries (Intersect and Difference):

= $(|Resp1|)Tp$

For Type-3 queries:

= $(|R1| + |R2|)Tp$

For Type-4 queries:

= 0

T5a = Time for Insert, Update, and Delete queries to modify the database in the Ms. Records are transmitted to the buffer via the RCBus and written into Ms

via MLink-2.

$$= (|\text{Resp1}|/n) (2\text{Trtrans} + \text{Ta} + \text{Tr})$$

The query processing time (FQP) is as follows:

For Type-1 queries:

Select and Project:

$$\begin{aligned} \text{FQP1a} = & \text{Tp} + \text{Tp}(2n+5) + \text{Tmtrans} + \\ & (1-Pm) [2\text{Tmtrans} + |\text{R1}|(\text{Ta} + \text{Tr} + \text{Trtrans})] + \\ & n\text{Tmtrans} + (|\text{R1}|/n)\text{Trtrans} + (|\text{Resp1}|/n)(\text{Tp} + \text{Trtrans}) + \\ & (|\text{Resp1}|)\text{Tp} \end{aligned}$$

Update and Delete:

$$\begin{aligned} \text{FQP1b} = & \text{Tp} + \text{Tp}(2n+5) + \text{Tmtrans} + \\ & (1-Pm) [2\text{Tmtrans} + |\text{R1}|(\text{Ta} + \text{Tr} + \text{Trtrans})] + \\ & n\text{Tmtrans} + (|\text{R1}|/n)\text{Trtrans} + (|\text{Resp1}|/n)(\text{Tp} + \text{Trtrans}) + \\ & (|\text{Resp1}|)\text{Tp} + (|\text{Resp1}|/n)(2\text{Trtrans} + \text{Ta} + \text{Tr}) \end{aligned}$$

For Type-2 queries:

Join:

$$\begin{aligned} \text{FQP2a} = & \text{Tp} + (n+6)\text{Tp} + \text{Tmtrans} + \\ & (1-Pm) [3\text{Tmtrans} + (|\text{R1}| + |\text{R2}|)(\text{Ta} + \text{Tr} + 2\text{Trtrans} + \text{Tc})] + \\ & \text{Tmtrans} + (|\text{R1}| + |\text{R2}|)\text{Trtrans} + \\ & (|\text{Resp1}| * |\text{Resp2}|)(2\text{Tp} + \text{Trtrans}) \end{aligned}$$

Intersect and Difference:

$$\begin{aligned} \text{FQP2b} = & \text{Tp} + (n+6)\text{Tp} + \text{Tmtrans} + \\ & (1-Pm) [3\text{Tmtrans} + (|\text{R1}| + |\text{R2}|)(\text{Ta} + \text{Tr} + 2\text{Trtrans} + \text{Tc})] + \\ & \text{Tmtrans} + (|\text{R1}| + |\text{R2}|)\text{Trtrans} + (|\text{Resp1}|)(2\text{Tp} + \text{Trtrans}) \end{aligned}$$

For Type-3 queries:

```
FQP3 = Tparse + (2n+5)Tp + Tmtrans +
       (1-Pm) [3Tmtrans+(|R1|+|R2|)(Ta+Tr+2Trtrans+Tc)] +
       Tmtrans + (|R1|+|R2|)(2Trtrans+2Tp)
```

For Type-4 queries:

```
FQP4 = Tparse + (2n+5)Tp +
       (1-Pm) [2Tmtrans+|R1|(Ta+Tr+Trtrans)] +
       n(Tmtrans) + (|R1|/n)Trtrans + (|Resp1|/n)Tp +
       (|Resp1|/n)(2Trtrans+Ta+Tr)
```

The complete F-Arch simulation model is presented in Appendix C. System components are modeled in the same manner as in the previous two models and the same three-phase modeling approach for component activity is used. Queries of all four types are generated and attributes set according to the workload model and experimentation plan.

Two-relation queries are processed by the IRP and records of the target relations are moved from Ms to the Mp, via the buffer and bus network. Once the records are available in Mp, the IRP accomplishes two-relation queries serially.

One-relation queries are processed simultaneously by the RUP network reading the records of the target relation from the buffer which is loaded from Ms. All the RUPs participate equally in processing queries. It is assumed that they can read records from the Ms and buffer structures in parallel. Statistics on

time in system (TIS) are collected for each query type as well as information about resource utilization for each system component.

The RRDS Architecture Model

The RRDS preliminary architecture components and query processing flow are illustrated in Figure 31. In this model, called P-Arch (Preliminary RRDS Architecture), requests are received from the host by the controller where they are parsed and transmitted to the RC network via the broadcast bus. All DBMS functions are accomplished by the RCs and a data partitioning scheme ensures equal participation by each processor in all queries. The RCs operate in parallel on their respective portions of the relational database. For one-relation queries, once the desired records are identified the appropriate operation is performed and the results transmitted to the controller. After receiving and compiling results from all the RCs, the controller sends the results to the host. Two-relation operations, though more complex in terms of processing, are accomplished in much the same manner and still maximize parallelism by utilizing the RCs to perform all data manipulation and processing functions. Once records of the target relations are identified by the RCs, each RC broadcasts its portion of one of the two relations involved in the operation to all the other RCs (this broadcast is not needed for the Union operation). Each RC then performs the desired operation between its portion of one relation and the entire second relation, and sends the results to the controller for compilation into a temporary relation and transmission to the

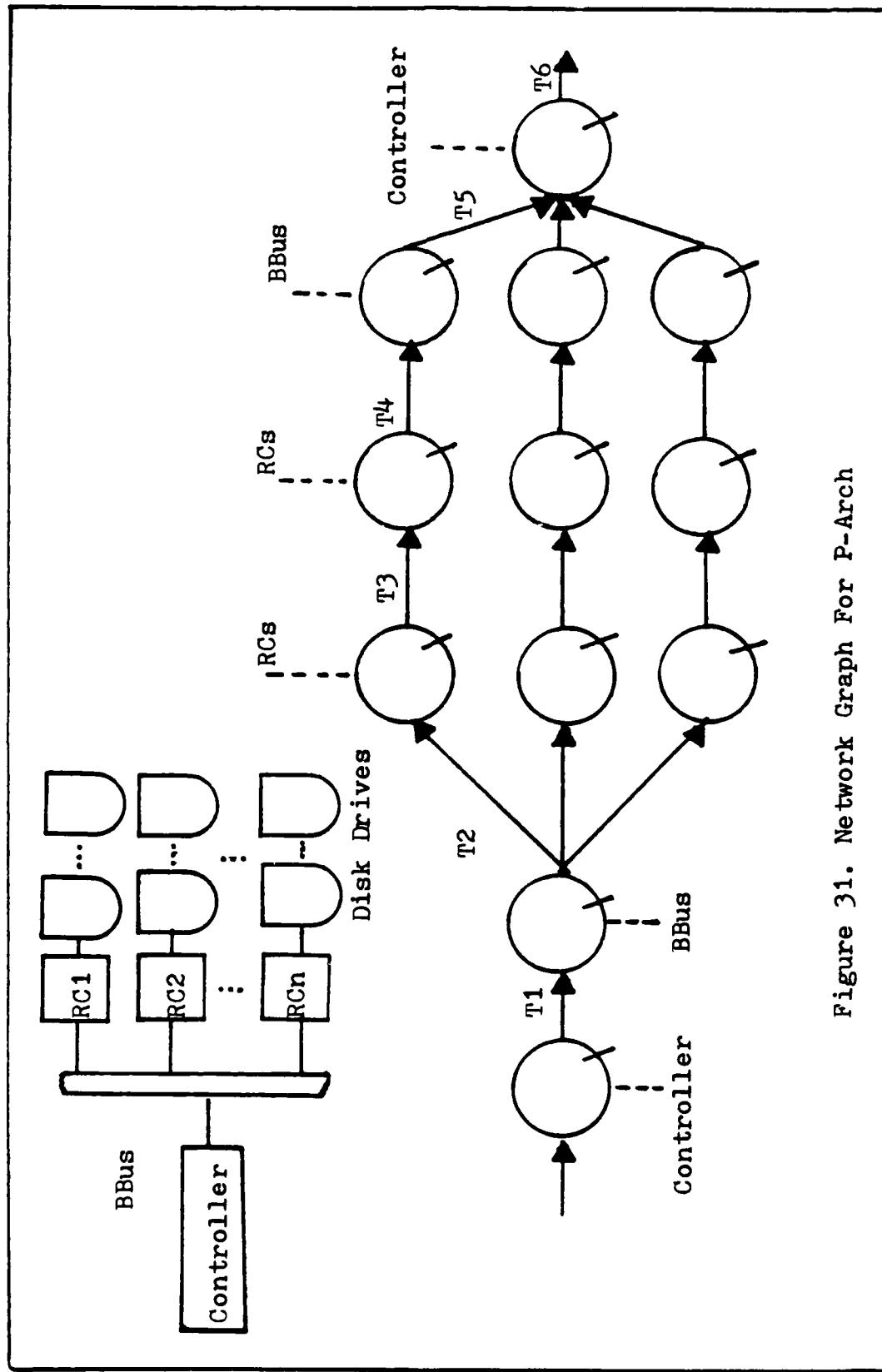


Figure 31. Network Graph For P-Arch

host.

We can now derive an expression for P-Arch query processing time (PQP).

$$PQP = \sum_{i=1}^6 T_i$$

Where:

T_1 = Time to parse query.

= T_{parse}

T_2 = Time to broadcast query to RCs.

= T_{mtrans}

T_3 = Time for each RC to locate and read the required relations off the disks.

For Type-1 queries:

= $[(|R_1|)T_a + (|Resp_1|)T_r]/n$

For Type-2 queries:

= $[(|R_1|+|R_2|)T_a + (|R_1|)T_r]/n$

For Type-3 queries:

= $[(|R_1|+|R_2|)(T_a + T_r)]/n$

For Type-4 queries, time to locate record position and insert.

= $[(|R_1|)T_a + (|Resp_1|)T_r]/n$

T_4 = Time for each RC to perform operation on records.

For Type-1 queries:

$$= (|\text{Resp1}|) T_p/n$$

For Type-2 queries, time to broadcast relation fragments and perform Join, Intersect, or Difference.

Join:

$$= (|\text{R2}|) T_{rtrans}/n + [(|\text{Resp1}|/n) * |\text{Resp2}|] T_p$$

Intersect and Difference:

$$= (|\text{R2}|) T_{rtrans}/n + (|\text{Resp1}|/n) T_p$$

For Type-3 queries:

$$= (|\text{R1}| + |\text{R2}|) T_p/n$$

For Type-4 queries, time to generate "operation complete" status message:

$$= T_p$$

T₅ = Time to send results to controller for collection into temporary relation and transmission to the host.

For Type-1 queries:

$$= (|\text{Resp1}|/n) T_{rtrans}$$

For Type-2 queries:

Join:

$$= [|\text{Resp1}|/n * |\text{Resp2}|] T_{rtrans}$$

Intersect and Difference:

$$= (|\text{Resp1}|/n) T_{rtrans}$$

For Type-3 queries:

= $[(|R1| + |R2|)/n]Trtrans$

For Type-4 queries, time to send status message to controller.

= $Tmtrans$

$T6 =$ Time to collect results into temporary relation.

For Type-1 queries:

= $(|Resp1|)Tp$

For Type-2 queries:

Join:

= $(|Resp1| * |Resp2|)Tp$

Intersect and Difference:

= $(|Resp1|)Tp$

For Type-3 queries:

= $(|R1| + |R2|)Tp$

For Type-4 queries:

= 0

So, the query processing time (PQP) is as follows:

For Type-1 queries:

$$PQP1 = Tparse + Tmtrans + [(|R1|)Ta + (|Resp1|)Tr]/n + \\ (|Resp1|)Tp/n + (|Resp1|/n)Trtrans + (|Resp1|)Tp$$

For Type-2 queries:

Join:

```

PQP2a = Tparse + Tmtrans + [(|R1|+|R2|)Ta + (|R1|)Tr]/n +
        (|R2|)Trtrans/n + [(|Resp1|/n)*|Resp2|]Tp +
        [(|Resp1|/n)*|Resp2|]Trtrans +
        (|Resp1|*|Resp2|)Tp

```

Intersect and Difference:

```

PQP2b = Tparse + Tmtrans + [(|R1|+|R2|)Ta +
        (|R1|)Tr]/n + (|R2|)Trtrans/n + (|Resp1|/n)Tp +
        (|Resp1|/n)Trtrans + (|Resp1|)Tp

```

For Type-3 queries:

```

PQP3 = Tparse + Tmtrans + [(|R1|+|R2|)(Ta+Tr)]/n +
        (|R1|+|R2|)Tp/n + [(|R1|+|R2|)/n]Trtrans +
        (|R1|+|R2|)Tp

```

For Type-4 queries:

```

PQP4 = Tparse + Tmtrans + [(|R1|)Ta + (|Resp1|)Tr]/n +
        Tp + Tmtrans

```

The complete P-Arch simulation model is presented in Appendix C. The approach to modeling components and activities is the same as the previous SLAM implementations. The query generation portion of the model consists of four Create nodes, each generating one of the four query types according to the workload model and experimentation plan. At generate time query attributes are assigned and the query is routed to the controller component. Parsing is modeled as a SLAM regular activity and the query proceeds to the bus component for broadcast to the RC

network. Simultaneous transmission is modeled as an n-way branch emanating from the bus component, with each branch terminating at its destination RC component. Activity at each RC component consists of locating and reading the appropriate records and performing the desired operation on them. For two-relation queries the broadcast of the smaller relation's fragments, to other RCs, also contributes to the RC component activity times. Once the processing is finished each RC awaits the bus to send results to the controller. Following transmission of the results, the controller must collect them into a temporary relation. Receipt of all the results by the controller is modeled using a Match node which is triggered when all the result fragments enter its queues. When triggered, on query identification attribute, the Match node routes the collection of results to the controller component for processing. Coalescing of the results is modeled as the final regular activity. Statistics on TIS and component utilization, for all four query types, are collected for analysis. In the next section, the workload model and experimentation plan, for the simulation models, are presented.

Workload Model and Experimentation Plan

Following development of the four architecture models, an experimentation plan was prepared. Each experiment was designed to test specific aspects of the architectures, providing information about their performance relative to the design questions of Chapter 3. To facilitate experimentation a workload

model was designed, allowing variation of database size, query mixes, and query arrival frequency.

The simulations were run for three database sizes-- small, medium, and large, and for each database size, three ratios were used-- $|R1|:|Resp1| = 3.3:1$, $|R2|:|Resp2| = 5:1$, and $|R1|:|R2| = 2:1$. The ability to vary database size facilitated identification of system bottlenecks in the processor and bus components by increasing the workload on these components during query processing and record transfer operations. Simulation runs were carried out on small databases to establish baseline performance for each model and to validate them against the analytical models.

The workload model also incorporates the capability to vary the query mix for each model to observe the effect of certain query types on each architecture. The ability to vary query mix is especially important in isolating the effects of functional specialization, and the effects of different query patterns, on response time. The design of the query generation portion of each simulation model allows great flexibility in altering query mix. The workload model was designed to allow testing configurations with an even query mix and with a preponderance of each query type. Query types modeled include Select, Join, Union, and Insert.

The ability to vary IAT facilitates imposition of heavy and light query arrival frequency on the system, simulating different

user behaviors. Increasing IAT also identifies bottlenecks as the arriving queries enter queues for system components. The complete workload model is presented in Table 4.

Six experiments were designed to test the behavior of the architectures across the spectrum of the workload model. A breakdown of the experiments along with an explanation of the objectives of each is provided in Table 5. Results of the experiments are presented in the next section.

A Comparison of The Four Architectures

TABLE 4. ARCHITECTURAL COMPARISON WORKLOAD MODEL

VARIABLE	RANGE	UNITS
Number of Queries	0 - 10	Queries
Mean IAT	5 - 60	Seconds
R1	100 - 10000	Records
R2	50 - 5000	Records
Resp1	30 - 3000	Records
Resp2	10 - 1000	Records
Query Mix (40 Queries)	Select Join Union Insert	
	25 5 5 5	Preponderance Select
	5 25 5 5	Preponderance Join
	5 5 25 5	Preponderance Union
	5 5 5 25	Preponderance Insert

TABLE 5. ARCHITECTURAL COMPARISON EXPERIMENTATION PLAN

EXPERIMENT	VARIABLES	GOAL
1	Query Type	Observe effect of fixed query type on small database providing baseline performance
2	p	Observe effect of degree of parallelism on performance of S-Arch
3	IAT Query Type	Observe effect of increasing IAT on response time
4	Query Mix	Observe impact of different mixes of queries on performance of architectures
5	n	Observe impact of adding more backend processors
6	Database Size n	Observe effect of increasing the size of database for a fixed backend system, and effect of increasing database size and adding more backends to each architecture

Table 6 shows the results of running each model for a fixed query type on a small database where relations consist of 100 fixed-length records. The RRDS architecture performed approximately three times better than each of the other systems for all query types. This is due to the fact that all of the RCs in RRDS operate in parallel on each query. In S-Arch response time is adversely affected by reduced parallelism and lack of a broadcast capability. The F-Arch and M-Arch models both

TABLE 6. FIXED QUERY - RESPONSE TIME

ARCHITECTURE	QUERY TYPE			
	SELECT	JOIN	UNION	INSERT
RRDS	.863	1.29	1.29	.861
S-Arch	2.62	n/a	n/a	2.63
M-Arch	2.95	4.41	4.01	2.98
F-Arch	2.57	4.20	4.20	2.59

exhibited bottleneck effects in the controller component and in the transfer of data from Ms to Mp components.

Under the SIMD approach of S-Arch parallelism is achieved when multiple queries, accessing different relations, are running on two or more backends simultaneously. The effect of the degree of parallelism on S-Arch response time is illustrated in Figure 32. For Select and Insert queries, arriving at a rate of one every 10 seconds, response time improved approximately 250% as the number of backends, operating in parallel, was increased from 1 to 3. Similarly, response time improved approximately 150% as the number of backends, operating in parallel, was increased from 3 to 6. Improvement was most dramatic for retrieval operations due to the fact that more processing was required, emphasizing the advantage of parallelism. These results indicate that S-Arch is a step towards extensibility, however, response time improvement will never be proportional to the number of additional backends as long as directory processing is

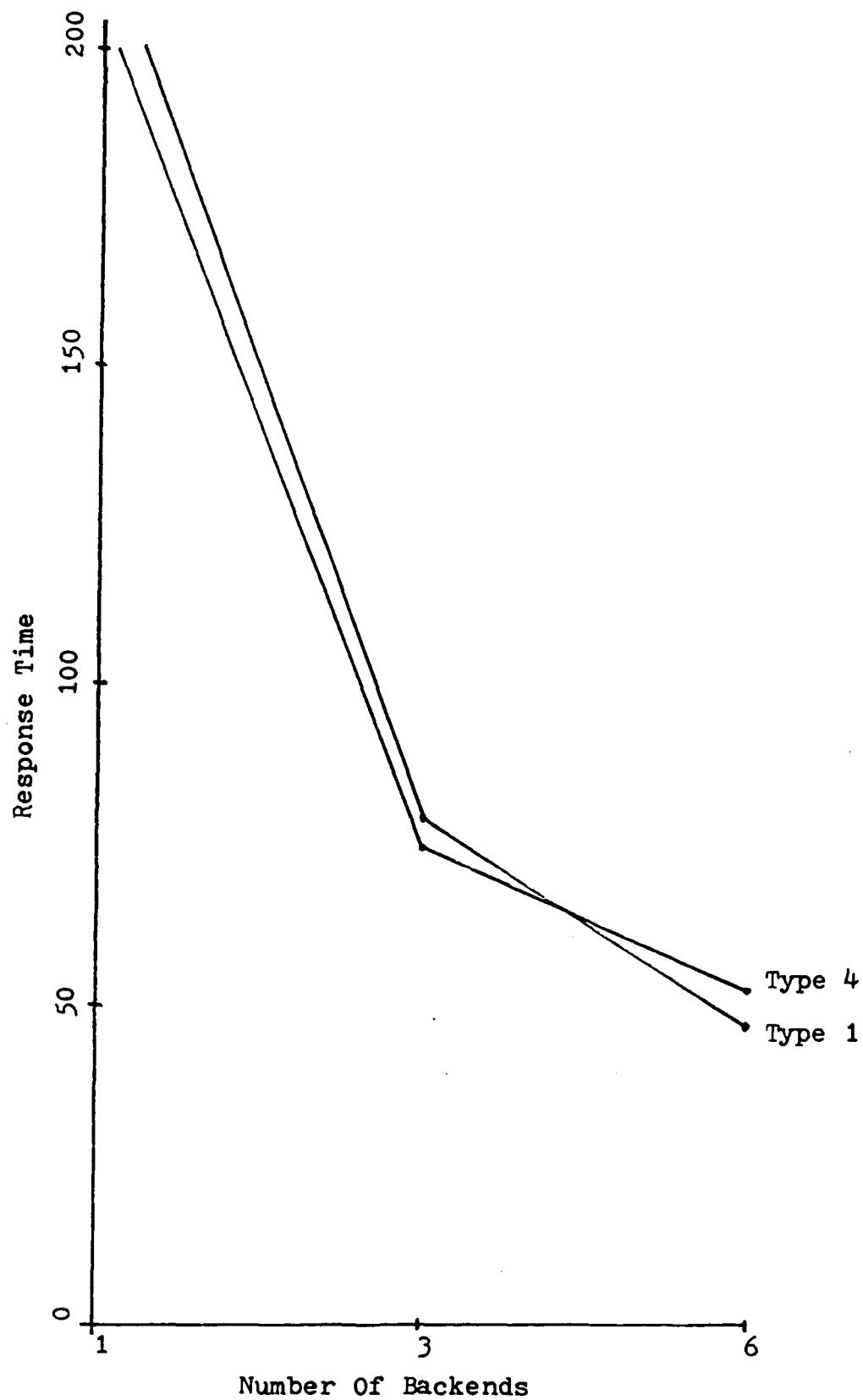


Figure 32. Effect Of Degree Of Parallelism On S-Arch Response Time

centralized (specialized backend problem) and messages are not broadcast (communications network limitation).

The adverse effect of controller limitation on performance of M-Arch and F-Arch became obvious when workload intensity was increased. Six-backend versions of M-Arch, F-Arch, and RRDS, processing databases with 1000 records per relation, were subjected to increased workloads as query interarrival times were decreased from 60 to 5 seconds. F-Arch and M-Arch performed poorly across the entire range of arrival times due to the heavy involvement of the controller in all phases of query processing. This controller limitation resulted in heavy queuing for the controller resources, and controller saturation, negating any benefits of the functional organization of F-Arch and parallelism in M-Arch. RRDS performed the best for all query types across the spectrum of interarrival times and showed the greatest sensitivity to variations in workload. The effect of increasing the query arrival frequency for each query type is illustrated in Table 7.

In an experiment to explore the effect of user behavior on the architectures supporting all query types (M-Arch, F-Arch, and RRDS) RRDS performed consistently better when subjected to various mixes of incoming query types. For architectures featuring six backend processors, operating on databases of relations with 1000 records each, queries of all four types were generated at a rate of one every 60 seconds. Four different workloads were imposed on each configuration, each representing a

TABLE 7. EFFECT OF QUERY INTERARRIVAL TIME ON RESPONSE TIME

QUERY INTERARRIVAL TIME

ARCHITECTURE	60	20	10	5
RRDS	6.48	28.04	61.77	81.35
M-Arch	1132.22	1296.00	1342.20	1364.56
F-Arch	1113.13	1269.70	1308.70	1328.60

Select Queries

QUERY INTERARRIVAL TIME

ARCHITECTURE	60	20	10	5
RRDS	14.46	91.96	173.53	193.15
M-Arch	1111.63	1300.70	1343.10	1362.67
F-Arch	1109.17	1266.20	1305.30	1325.20

Join Queries

QUERY INTERARRIVAL TIME

ARCHITECTURE	60	20	10	5
RRDS	9.06	29.08	68.10	91.28
M-Arch	1133.40	1309.00	1342.60	1364.56
F-Arch	1066.92	1252.50	1298.40	1321.80

Union Queries

QUERY INTERARRIVAL TIME

ARCHITECTURE	60	20	10	5
RRDS	7.58	26.64	63.98	88.66
M-Arch	1164.90	1284.30	1337.00	1359.11
F-Arch	1052.79	1249.60	1298.60	1323.50

Insert Queries

preponderance of one query type. As shown in Table 8, RRDS performed well regardless of the particular query mix, a desirable trait in an architecture designed for general database

applications. Two-relation queries (Join and Union), requiring more processing, caused queuing to occur for major system resources in M-Arch and F-Arch, further exacerbating the controller limitation problem in these architectures. In F-Arch, controller and channel limitation resulted in such severe queuing for the controller and bus network that the advantages of functional division were lost.

All four architectures, as expected, were highly sensitive to increases in the size of the database, however, RRDS again exhibited superior performance. Select queries were run on architectures with six parallel processors and the size of the database was ranged from 100 records/relation to 10000 records/relation. As shown in Table 9, the simplest architectures, S-Arch and RRDS, showed less response time degradation as the size of the database was increased. The poor performance of M-Arch and F-Arch is attributed to queuing due to the heavy involvement of the controller in all actions and data

TABLE 8. EFFECT ON RESPONSE TIME OF DIFFERENT QUERY MIXES

DOMINATING QUERY TYPE

ARCHITECTURE	SELECT	JOIN	UNION	INSERT
RRDS	7.80	9.75	8.98	7.72
M-Arch	707.40	958.70	975.30	700.60
F-Arch	847.15	1225.73	1199.61	646.40

TABLE 9. INCREASED DATABASE SIZE - RESPONSE TIME

ARCHITECTURE	RELATION SIZE (RECORDS)		
	100	1000	10000
RRDS	.49	4.38	68.29
S-Arch	2.73	27.28	336.00
M-Arch	3.14	60.29	2519.10
F-Arch	2.77	48.80	2290.10

transfer overhead.

Finally, as shown in Figure 33, RRDS shows the greatest hope for extensibility. For Select queries, on each of the four architectures, the size of the database was increased from 1000 records/relation to 3000 records/relation in increments of 1000 records/relation. Simultaneously, the number of backend processors in each system was increased from three to nine in increments of three. M-Arch and F-Arch experienced the worst degradation due to the fact that controller overhead and data transfer overhead dominated the benefits of increased parallelism realized by adding backends. S-Arch also performed poorly due to the fact that directory management for all queries is centralized, causing queuing at the directory backend. RRDS showed only a minute increase in response time as the increased parallelism of adding more RCs offset the increase in system workload.

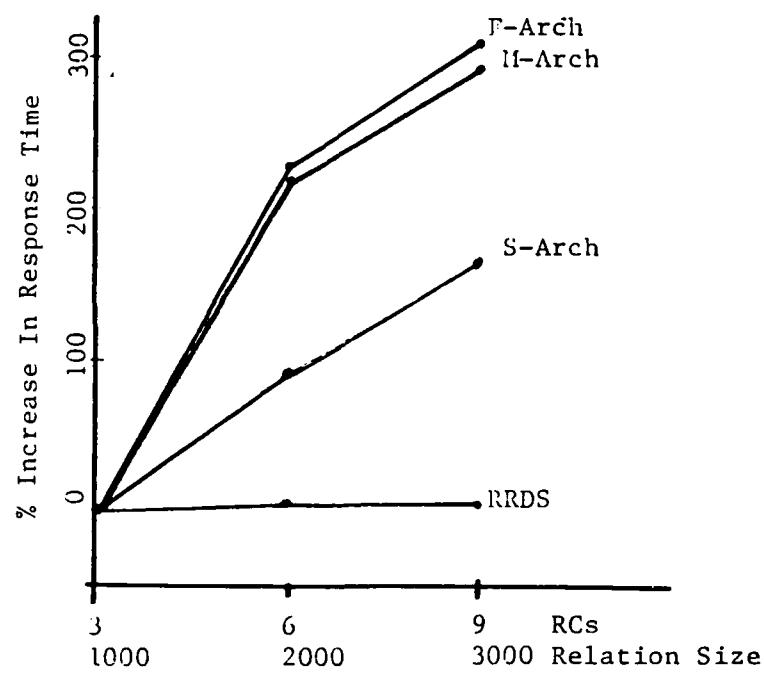


Figure 33. Increased Database Size And Increased Number Of Backends - Percent Increase In Response Time

The results of the comparative study indicate that the simpler approach of RRDS will perform the best in the general case, and the goals of maximum parallelism and an equally processed and partitioned database and directory are valid. In addition, the need to minimize controller involvement, in query processing, was emphasized.

CHAPTER 5

DATA ACCESS IN RRDS

In the preceding chapter a generic multi-backend architecture was refined into a preliminary hardware configuration for RRDS. In this, and the next two chapters, phases two through four of the design process, addressing software questions, are presented. Software considerations include selecting a data access strategy, a data placement strategy, and a directory management strategy for the system.

In this chapter a data access strategy is selected for RRDS. The strategy selected must be compatible with the replicated computer and partitioned database approach, and must minimize controller involvement in query processing. In addition, the strategy must facilitate efficient range query processing in a large database environment. The most important selection criterion is response time, however, the resulting directory size is also a crucial factor since very large databases could result in very large directories.

Three data access strategies--hashing, hierarchical indexing, and clustering were considered for RRDS. Although an extremely fast method for look up on a primary key, hashing performs poorly for range queries since it cannot access records easily in order of sorted key values. For this reason hashing

was eliminated from further consideration. In the rest of this chapter the remaining two strategies will be studied in depth. We develop a clustering scheme and a hierarchical indexing scheme, based upon the B+_Tree data structure, for RRDS. Analytical and simulation models are developed for query processing under each approach and evaluated with respect to the performance criteria discussed above. The best approach for RRDS is determined and integrated into the system.

A Clustered Data Access Strategy For RRDS

The goal of a clustered database is to reduce data access time by isolating likely response sets. Each relation is partitioned into a non-intersecting set of clusters which are, in turn, spread across the system according to some data placement policy. Advantages of this method include 1) reducing the segment of a relation that needs to be processed to answer a query, and 2) providing relation subsets for distribution across the system, facilitating parallel processing. Disadvantages include replicated directories and cluster creation and management overhead.

The clustering scheme developed for RRDS is similar to that of MDBS (Hsiao 1981a), albeit simpler due to the fact that RRDS clustering is based upon relations instead of attribute-value pairs which characterize the attribute-based data model of MDBS.

Clustering Concepts and Terminology

Each relation in RRDS would consist of a set of non-intersecting clusters, each cluster containing a set of records, each record having all of the attributes defining the relation scheme. The records in a relation are grouped into clusters based on attribute values and attribute-value ranges. These values and value ranges are called descriptors.

Descriptors can be defined in terms of predicates. A predicate is of the form: (<attribute><relational operator><value>) where the relational operator is from the set {<, <=, >, >=, <>, =}. Descriptors can be of two types. The first, called a range descriptor, is a conjunction of a greater-than-or-equal-to predicate and a less-than-or-equal-to predicate such that the same attribute appears in both predicates, i.e., ((attribute >= value1) AND (attribute <= value2)) which can also be written as (value1 <= attribute <= value2). The second type descriptor, called an equality descriptor, consists of a simple equality predicate, i.e., (attribute = value). Ranges specified in range descriptors, for a given attribute, must be mutually exclusive, and for every equality descriptor no range descriptor can have the same attribute and a range containing the same value.

The attributes for which descriptors are defined are called directory attributes and are the relation keys. Descriptors are defined by the database creator and used by the system to form

clusters. A cluster is simply a set of similar records to be retrieved together--a likely response set. For example, given the relation scheme:

```
EMPLOYEES(EMP#, NAME, DEPT, SALARY)
```

one cluster for this relation might contain records for employees in the Toy department who earn between \$15,000 and \$30,000. Thus records of this cluster are grouped by the following descriptors:

```
{(DEPT = Toy), (15000 <= SALARY <= 30000)}
```

Queries are expressed in terms of conjunctions (specifiers). A query conjunction is a conjunction of predicates, e.g.,

```
(DEPT = Toy) AND (SALARY < 20000)
```

A record satisfies a conjunction if it contains attribute values that satisfy every predicate in the conjunction.

Data Access Based Upon a Clustering Scheme

In this section we define a clustering scheme for RRDS in terms of required data structures and algorithms. For a relation, the database creator specifies a group of descriptors for clustering purposes. Attributes that appear in the descriptors are called directory attributes. A record consists of values for the attributes comprising the relation scheme. Only values for directory attributes in a record are considered for clustering purposes. From the rules of descriptors stated in the previous section, it can be seen that a directory attribute

value is derivable from, at most, one descriptor, hence a record is derived from a set of descriptors. If two records are derived from the same set of descriptors they are likely to be retrieved together, in response to a query, and therefore belong to the same cluster. Hence, a cluster is a subset of a relation, a group of records such that every record in the cluster is derived from the same descriptor set. A record cluster is defined by the set of descriptors from which all the records in the cluster are derived. One can see that a record belongs to one, and only one, cluster. Each relation scheme has one, or more, directory attributes. Each record in a relation, then, must be derived from one, and only one, set of descriptors since each value of a directory attribute is derived from, at most, one descriptor. The set of descriptors for a record defines the cluster to which the record belongs. Thus, the concept of descriptor sets has been used to partition each relation into equivalence classes called clusters.

Since the clusters represent likely response sets each cluster can be spread across the RC network, of RRDS, so that all RCs participate equally in retrieval operations, the main benefit of this technique. Let us present an example to illustrate the RRDS clustering technique.

Given the relation:

EMPLOYEES					
RECORD ID	EMP #	NAME	DEPT	SALARY	
r1	1001	Jones	Toy	10000	
r2	1002	Smith	Pet	22000	
r3	1003	Hanson	Toy	13000	
r4	1004	Harris	Toy	19000	
r5	1005	Elder	Pet	22000	
r6	1006	Jackson	Pet	30000	
r7	1007	Morris	Toy	10000	
r8	1008	Abel	Toy	19000	
r9	1009	Rice	Pet	36000	
r10	1010	Schaper	Toy	18000	
r11	1011	Baxter	Pet	29000	
r12	1012	Harper	Pet	21000	

with directory attributes DEPT and SALARY, and descriptors defined as follows:

D1: (DEPT = Toy)

D2: (DEPT = Pet)

D3: (0 <= SALARY <= 20000)

D4: (20001 <= SALARY <= 50000)

The relation is partitioned into two clusters as follows:

Cluster 1:

Descriptor Set: {D1, D3}

Records : {r1, r3, r4, r7, r8, r10}

Cluster 2:

Descriptor Set: {D2,D4}

Records : {r2, r5, r6, r9, r11, r12}

(Note that insertion of new records into the relation may create the cluster with descriptor set {D1, D4} and the cluster with descriptor set {D2, D3}.)

Since each cluster is a likely response set, if the records in a cluster are evenly distributed across the RCs of the system, as illustrated in Figure 34, retrieval operations will utilize all RCs evenly in parallel. For example, the query:

Select All From EMPLOYEES

Where

(DEPT = Toy)

will result in selection of records r1, r4, and r8 from RC1 and records r3, r7, and r10 from RC2.

Data Structures to Support Clustering. Creating and maintaining clusters, in RRDS, requires a set of cluster tables comprising the bulk of the system directory as well as a set of algorithms for utilizing the tables. The cluster directory must be replicated at each RC, and each directory table is stored in the primary (Mp) or secondary memory (Ms) component of the RC depending upon the table size. Since these structures are replicated, and may become voluminous, we will be interested in the impact of their size, versus the size of structures for the hierarchical indexing technique, during the analysis of these two data access strategies.

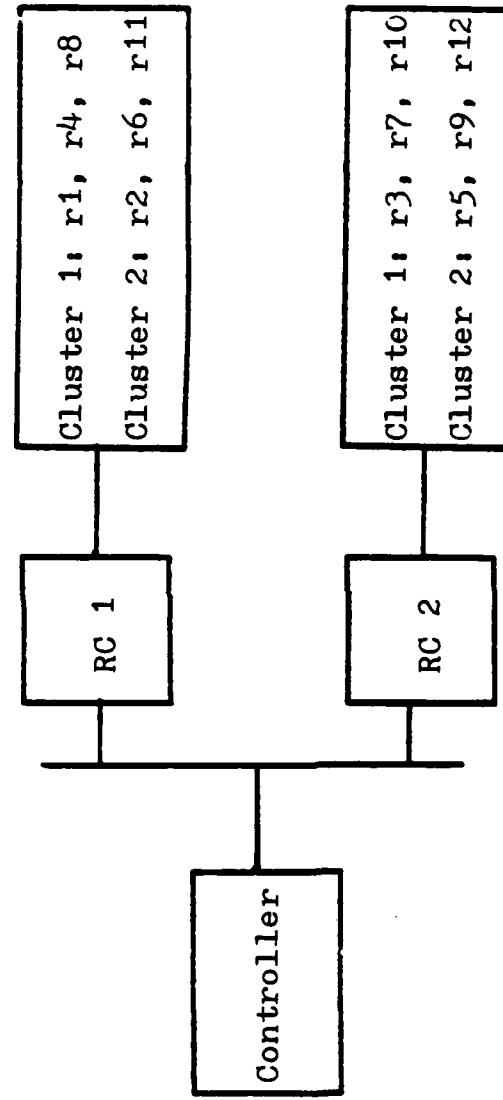
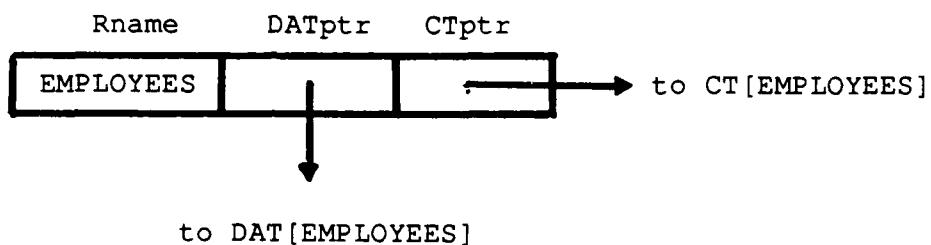


Figure 34. Distribution Of EMPLOYEES Across A Two-RC System

A relation information table (RIT) is required to associate each relation with its corresponding directory attribute table (DAT) and cluster table (CT). For each directory attribute there is a list of descriptors maintained in a descriptor table (DT).

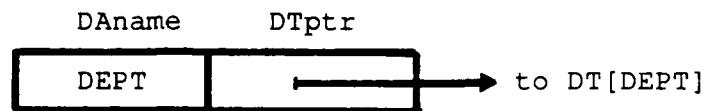
The RIT contains information on each relation in the database. This is a relatively small table, in terms of number of entries, and is therefore located in the primary memory of the RC. Each entry in the RIT consists of the name of a relation and a pair of pointers as shown below:



The first pointer provides access to the DAT for the relation and the second provides access to the relation's CT. The RIT is the first structure consulted by the clustering algorithms in order to locate other cluster directory tables. There is one RIT for the database.

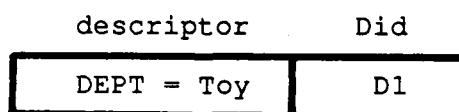
For each relation in the database there is a DAT stored in primary memory. This structure is also small containing an entry for each directory attribute in the relation. Each entry consists of a directory attribute name and a pointer providing access

to the DT for the directory attribute. An example DAT entry is shown below:



Cluster management algorithms use this table to locate the descriptor table associated with each directory attribute. There is one DAT for each relation in the database.

For each directory attribute there is a set of descriptors stored in a DT, located in secondary memory. The size of the DT is governed by the number of descriptors the database creator deems appropriate for the given directory attribute, and can range from: no entries to an entry (descriptor) for each value of the directory attribute. Each DT entry consists of a descriptor and a descriptor identifier (Did), for example:



The clusters comprising a relation, in the database, are described by a CT. This structure, which may become large, is stored in the secondary memory. Each CT entry is made up of three parts. The first, a cluster identifier (Cid), is the unique name of the cluster. Next, the CT entry contains the

descriptor set (Dset) describing the cluster. Finally, each CT entry contains the addresses of the records, in the RC's secondary memory, which belong to the cluster. The CT is the basic cluster directory accessed by the cluster management algorithms in all query processing. The size of the CT is highly dependent upon how the relation is partitioned by the database creator. The number of entries can range from: one--the case where all of the records in the relation are in one cluster, to r , where r is the number of records in the relation--the case where each record is a separate cluster. During the simulation study the impact of these alternative partitioning approaches on system response time will be examined. An example CT entry is provided below:

Cid	Dset	addr
C1	{D1,D3}	r1, r4, r8

Figure 35 illustrates the clustering data structures, and their relationships, for the EMPLOYEES relation example.

Having defined the data structures required to support the RRDS clustering scheme, expressions for their size may now be presented. Space requirements for these structures are especially important because they must be replicated at the RCs. First, a set of variables is defined forming the basis for the model. Next, the expression for the size of each structure is given in terms of the variable set. The size of the RIT is

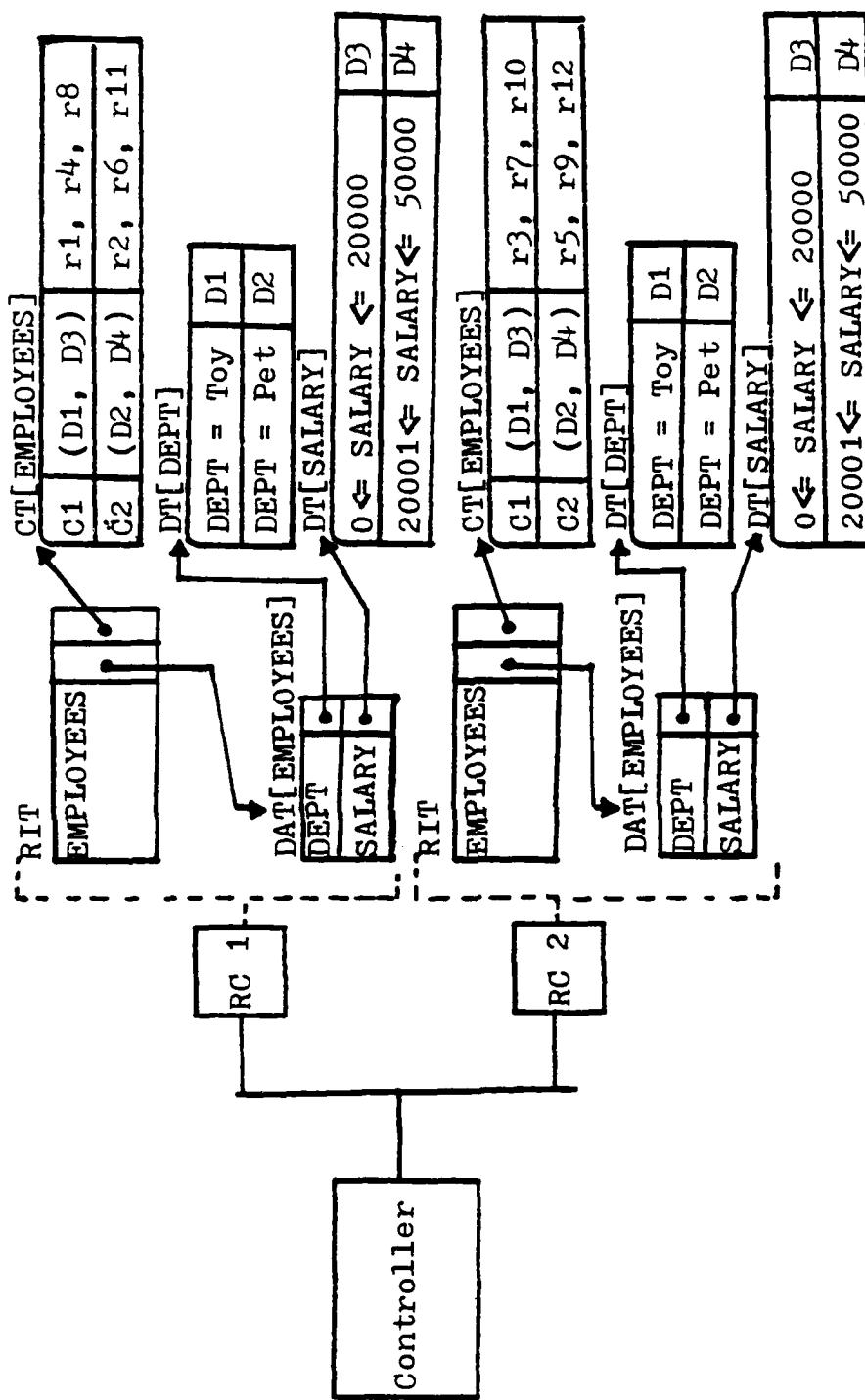


Figure 35. Clustering Data Structures For EMPLOYEES Relation

simply a function of the number of relations comprising the database. The sizes of the DAT, DT, and CT, however, depend heavily upon how the creator partitions a relation. For this reason we present the expressions, for each of these structures, in terms of smallest, largest, and the general case.

The following variables are used in developing the clustering table size analytical model:

<u>Variable</u>	<u>Description</u>
RC	Number of RCs in RRDS system
R	Number of relations in database
Rname	Average size of relation name (bytes)
ptr	Size of pointer (bytes)
r	Number of records in a relation
a	Number of attributes for a relation
da	Number of directory attributes for a relation ($1 \leq da \leq a$)
a	Average size of attribute name (bytes)
d	Number of descriptors for a directory attribute
d	Average descriptor size (bytes)
did	Average descriptor identifier size (bytes)
cl	Number of clusters in a relation
rcl	Number of records in a cluster
cid	Average size of cluster identifier (bytes)
dset	Number of descriptors in a descriptor set Note that, since all the records in a relation have the same attributes (hence the same number of directory attributes), all the records in a

relation have the same number of descriptors in their descriptor sets.

addr	Average size of record address (bytes)
RIT	Size of RIT (bytes)
DAT	Size of DAT (bytes)
DT	Size of DT (bytes)
CT	Size of CT (bytes)

The expressions for the sizes of the clustering data structures are as follows:

Structure: RIT Location: RC primary memory

Number of RITs at each RC: 1

Size:

Number entries: R
Entry size : Rname + 2(ptr)

| RIT | = R(Rname + 2(ptr))

Structure: DAT[R] Location: RC primary memory

Number of DATs at each RC: R

Size:

Number entries: da
Entry size : |a| + ptr

Smallest	$ DAT[R] = a + ptr$
Largest	$ DAT[R] = a(a + ptr)$
General	$ DAT[R] = da(a + ptr)$

Structure: DT[da] Location: RC secondary memory

Number of DTs at each RC: $da[R1]+da[R2]+\dots+da[Rn]$, $1 \leq n \leq R$

Size:

Number of entries: d
 Entry size : $|d| + did$

Smallest $|DT[da]| = |d| + did$
 Largest $|DT[da]| = r(|d| + did)$
 General $|DT[da]| = d(|d| + did)$

Structure: CT[R]

Location: RC secondary memory

Number of CTs at each RC: R

Size:

Number of entries: cl
 Entry size : Depends upon size of dset for each
 cluster and number of records in
 each cluster

Smallest: $|CT[R]| = cid + did + r/RC(addr) \Rightarrow$ One entry
 in CT with one descriptor in dset and all
 records in one cluster.

Largest: $|CT[R]| = (r/RC)[cid + a(did) + addr] \Rightarrow$ One
 record per cluster and all attributes defined
 as directory attributes.

General: $cl[cid + (da)(did) + (rcl)(addr)]$

Now that the directory structures have been defined, a set of algorithms for managing and maintaining the clusters is presented.

Algorithms For RRDS Clustering Scheme Directory Management. Query processing, in a clustered RRDS, requires some directory management overhead. The two main actions include operations for

record insertion and retrieval. Insertion operations require cluster creation and maintenance actions on the directory. Queries involving data retrieval require consultation of the directory data structures to determine which clusters contain the target records, as well as the locations of the records on secondary storage. In this section the directory management algorithms are presented, for the RRDS clustering scheme, based upon the data structures previously defined. Cluster management and maintenance algorithms are provided for record insertion (Insert), record retrieval (Select), and the two-relation operation Join. For each algorithm an explanation is provided, followed by the algorithm stated in pseudo-code. Next, new analytical parameters and variables are defined followed by derivation of expressions for directory management times for each algorithm. These expressions are later used to construct simulation models, to determine the impact of cluster management overhead, and for comparison with the hierarchical index strategy.

The cluster insertion algorithm applies to insertion of records into relations. A record, to be inserted into the database, is broadcast to the RCs. The RC (determined by data placement strategy discussed in the next chapter) which will actually insert the record must consult its RIT to find the target relation entry. The RIT provides access to the remaining directory structures--the DAT, DT, and CT. The attribute fields of the record are searched to determine which ones are directory

attributes according to the DAT for the relation. For each directory attribute the DT is consulted and the value associated with the attribute is compared against the descriptors in DT. When a descriptor is found, which covers the value, its descriptor identifier (Did) is added to a set of Dids, being compiled for the record, called the record descriptor set (Rdset).

Once all the values for the directory attribute fields have been compared with the descriptors in appropriate DTs, the RIT is consulted to access the CT. For each entry in the target relation's CT, the descriptor set for the record is compared with the descriptor set (Dset) for the cluster. If a match is found then the record is inserted into the secondary storage and its address added to the list of addresses for the cluster. If a matching cluster is not found for record insertion then a new cluster is formed with its descriptor set equal to Rdset. The record insertion and cluster creation algorithm is presented below:

Record Insertion and Cluster Creation Algorithm

Input: A record for insertion into a relation in the RRDS database, of the form:

attr[1] attr[2] ... attr[a]
r =

value1	value2	...	value a
--------	--------	-----	---------

BEGIN

 Consult RIT to find target relation entry.

```

FOR each record, r, to be inserted in relation R DO
(* Find the descriptor set for record to be inserted *)
  FOR each attribute field attr[1]..attr[a], in r, DO
    Consult the DAT.
    IF attr[i] is a directory attribute THEN
      Consult the DT.
      FOR descriptors in DT, until a match is found, DO
        IF the value v[i] of attr[i] is derived
        from this descriptor THEN
          Add this Did to the Rdset.
        ENDIF
      ENDFOR
    ENDIF
  ENDFOR
ENDFOR
(* Find the cluster for the record to be inserted *)
Consult the CT.
FOR each cluster entry in CT, until a match is found, DO
  IF descriptor set for the record is equal to descriptor
  set for the cluster THEN
    Add the record to the cluster.
  ENDIF
ENDFOR
IF record not in an existing cluster THEN
  Create a new cluster with descriptor set equal to
  descriptor set of the record.
ENDIF
ENDFOR
END

```

Output: The new record's address added to the appropriate cluster, or the creation of a new cluster and accompanying modification of CT for target relation.

The following new parameters and variables are required for formulation of an expression for record insertion directory management time:

Parameter	Description
Trit	Time to access an entry of RIT from primary memory
Tdat	Time to access an entry of DAT from primary memory
Tdt	Time to access an entry of DT from secondary memory

Tdid	Time to read a descriptor identifier
Taddr	Time to read/write a record address
<u>Variable</u>	<u>Description</u>
RIT	Number of entries in RIT
DAT	Number of entries in a relation's DAT
DT	Number of entries in a directory attribute's DT
CT	Number of entries in a relation's CT
Pda	Probability that an attribute is a directory attribute
Pnc	Probability that the record to be inserted does not belong to an already existing cluster (probability of new cluster)

The expression for time to accomplish directory management for an insert query (TDMI) is now derived. This time consists of the summation of the times to access and/or modify each of the directory data structures.

$$TDMI = T1 + T2 + T3 + T4$$

where:

T1 = Time to access RIT to find target relation entry

= $RIT/2(Trit)$: assuming on the average half of the RIT must be searched to find the relation entry

T2 = Time to access DAT to determine directory attributes

= $a(DAT/2)(Tdat)$: assuming on the average half of the DAT must be searched to determine if an attribute in the record is a directory attribute

T3 = Time to access DTs to determine descriptors involved
= da(DT/2) (Tdt) : assuming on the average half of the
DT must be searched to find the
descriptor covering the directory
attribute value

T4 = Time to access CT and insert into existing cluster
or create a new cluster entry
= (1-Pnc){(CT/2)[dset(Tdid)] + Taddr + Taddr} +
Pnc{(CT)[dset(Tdid)] + dset(Tdid) + Taddr}
Assumption: If the record belongs to an already
existing cluster on the average half of the CT
must be searched.

Clustered directory management for data retrieval operations (SELECT queries) is a complex three-step process. In the first phase (descriptor search) a set of descriptors is found for each predicate in the query. First, the RIT is consulted to find the target relation entry and related directory data structures. For each predicate of the query, containing a directory attribute, the DT associated with the directory attribute is accessed and the descriptor(s) covering the predicate is located and added to the predicate descriptor set (Pdset). For each query conjunction the Pdsets are combined to form a conjunction descriptor set (Cdset). The output of the first phase, the query descriptor set (Qdset), consists of all the Cdsets.

In the second phase (cluster search) the Pdsets are used to determine a set of clusters satisfying the query. For each query conjunction a set of descriptor groups (Dgroups) is formed from

the cartesian product of the Pdsets for the predicates in the conjunction (i.e., Dgroup is an element of the cross product of Pdset[i] where $1 \leq i \leq p$ and p is the number of predicates in the conjunction). The Dgroups represent the possible combinations of descriptors satisfying the query conjunctions. For each entry in the CT each Dgroup is compared with the cluster's descriptor set (Dset). Clusters, whose Dset contain at least one of the Dgroups, are added to the set of clusters satisfying the query, called the query cluster set (Qclset).

Once the clusters satisfying the query have been determined the third phase (address generation) can commence. In this phase the clusters of the Qclset are located, in the relation's CT, and the record addresses associated with each cluster are read. The address generation phase constitutes a separate step from the cluster search phase because, even though the CT entries are available from the cluster search phase, security and concurrency control activity must be accomplished prior to actually granting access to the clusters. The cluster directory management algorithm for retrieval operations is presented below:

Cluster Directory Management Algorithm For Data Retrieval

Input: Data retrieval query consisting of c conjunctions where the number of predicates in a conjunction is p (assuming all predicates are on directory attributes).

```
BEGIN (* phase 1: descriptor search - construct Qdset *)
  Consult RIT to find entry for relation R.
  FOR each conjunction in query DO
    FOR each predicate in conjunction DO
```

```
IF relop is "=" THEN
    Access the DT for the attribute (in the predicate)
    and search to find the descriptor satisfying the
    predicate.
    Add Did of descriptor to Pdset.
ELSE (* relop is not "=" *)
    Access DT for the attribute (in the predicate)
    and search to find all descriptors which
    satisfy the predicate.
    Add Dids of descriptors to Pdset.
ENDIF
    Add the Pdset to the Cdset.
ENDFOR
ENDFOR

(* end phase 1: Output is a Qdset: *)  

(* Qdset = {Cdset[1], Cdset[2] ,..., Cdset[c]} *)  

(* Cdset[i] = {Pdset[1], Pdset[2] ,..., Pdset[p]} *)  

(* begin phase 2: cluster search - construct Qclset *)
(* input: Qdset *)
FOR each conjunction in query DO
    Form Dgroup(s) where Dgroup is in cross product
    of Pdset[i] : 1 <= i <= p.
    (i.e., Pdset[1] X Pdset[2] X ... X Pdset[p])
ENDFOR
FOR each entry in CT DO
    FOR each Dgroup DO
        IF Dgroup is a subset of Dset for the CT entry THEN
            Add the Cid to the Qclset.
        ENDIF
    ENDFOR
ENDFOR

(* end phase 2: output is a Qclset consisting of Cids *)

(* begin phase 3: address generation *)
(* input: Qclset *)
FOR each entry in CT DO
    IF an element of the Qclset = Cid THEN
        Read all record addresses for the cluster.
    ENDIF
ENDFOR

(* end phase 3 *)

END

Output: Addresses for records satisfying the
data retrieval query.
```

The following new parameters and variables are required for formulation of an expression for cluster directory management for data retrieval:

<u>Parameter</u>	<u>Description</u>
Tdset	Average time to read a Dset from CT
Tcomp	Time to compare Dgroup and Dset
Tcc	Time to generate a Dgroup
<u>Variable</u>	<u>Description</u>
TQdset	Time to construct Qdset
TQclset	Time to construct Qclset
Tadgen	Time for address generation
pdset	Average size of pdset (number of Dids)
c	Number of conjunctions in query
p	Number of predicates in a conjunction
Peq	Probability that predicate's relop is "="
crec	Average number of records per cluster
qclset	Average size of Qclset (clusters)

The expression for time to accomplish directory management for a data retrieval query (TDMS) is now derived. This expression consists of the times to accomplish the three phases of the algorithm.

TDMS = TQdset + TQclset + Tadgen

$$TQdset = RIT/2(Trit) + \sum_{i=1}^C p[i] [P_{eq}(DT/2) Tdt + (1-P_{eq})(DT) Tdt]$$

Assumptions:

- 1) Time to read and transfer a descriptor (of DT) from Ms dwarfs time to compare the descriptor with the predicate.
- 2) Equality predicate requires, on the average, search of half of the DT.

$$TQclset = \left[\sum_{i=1}^C \left[\prod_{j=1}^p pdset[j] \right] (Tcc) \right] + CT[Tdset + \sum_{i=1}^C \left[\prod_{j=1}^p pdset[j] \right] (Tcomp)]$$

Tadgen = (qclset)(crec)(Taddr)

Cluster directory management for the two-relation operations such as JOIN are considerably simpler. These operations are based upon all the records of the target relations instead of a subset, i.e., there is no specifier in the query. For this reason, all of the clusters comprising the two relations are accessed via the CTs and all record addresses read. First, as before, the RIT is consulted to find pointers to the CTs of both target relations. The CT for each relation is then accessed, and the record addresses for each of the clusters are read. The cluster directory management algorithm for two-relation operations is presented below:

Cluster Directory Management Algorithm For Two-Relation Operations

Input: A two-relation query

```
BEGIN
    Consult RIT to find entry for R1.
    FOR each entry in CT[R1] DO
        Read all record addresses.
    ENDFOR
    Consult RIT to find entry for R2.
    FOR each entry in CT[R2] DO
        Read all record addresses.
    ENDFOR
END
```

Output: Addresses of all records in R1 and all records in R2

The expression for time to accomplish directory management for a two-relation operation (TDMJ) is as follows:

$$TDMJ = 2(RIT/2)Trit + [(cl[R1])(crec)Taddr +
 (cl[R2])(crec)Taddr]$$

A Hierarchical Indexing Scheme Based Upon B+ Trees

An alternative data access strategy for RRDS would use B+_trees as hierarchical indices to the relations. B_trees have become a standard file organization and their characteristics and advantages are well documented in the literature (Comer 1979; Ullman 1982; Horowitz 1982). Implementing an indexing scheme, instead of clustering, results in a smaller and non-replicated directory. The entire relation is now distributed across the RCs

and index structures are maintained on the relation partition residing at each RC. The indexed data access scheme developed for RRDS is based upon $B+$ _trees. In the following sections this strategy is explained in terms of required data structures and algorithms. As in the development of the clustering data access strategy, expressions will be derived for both the memory requirements (data structure size) and directory management time for query processing. The last part of this chapter details a simulation study of the two strategies based upon these two criteria.

B+Tree Concepts and Terminology

Due to the abundance of literature concerning B-trees, and their variations, we will restrict ourselves to a brief discussion of $B+$ _tree characteristics and terminology. In a $B+$ _tree all search keys are located in the leaves and upper levels, organized as a B-tree, constitute the index. This structure is ideally suited for range queries due to the fact that the leaves are linked in sequential order. This linked list of leaves is called the sequence set (Comer 1979). Figure 36 illustrates the general format of a $B+$ _tree. These data structures maintain their efficiency due to the fact that they remain balanced despite insertions and deletions. This balancing process could constitute the major overhead in directory management under this indexing scheme.

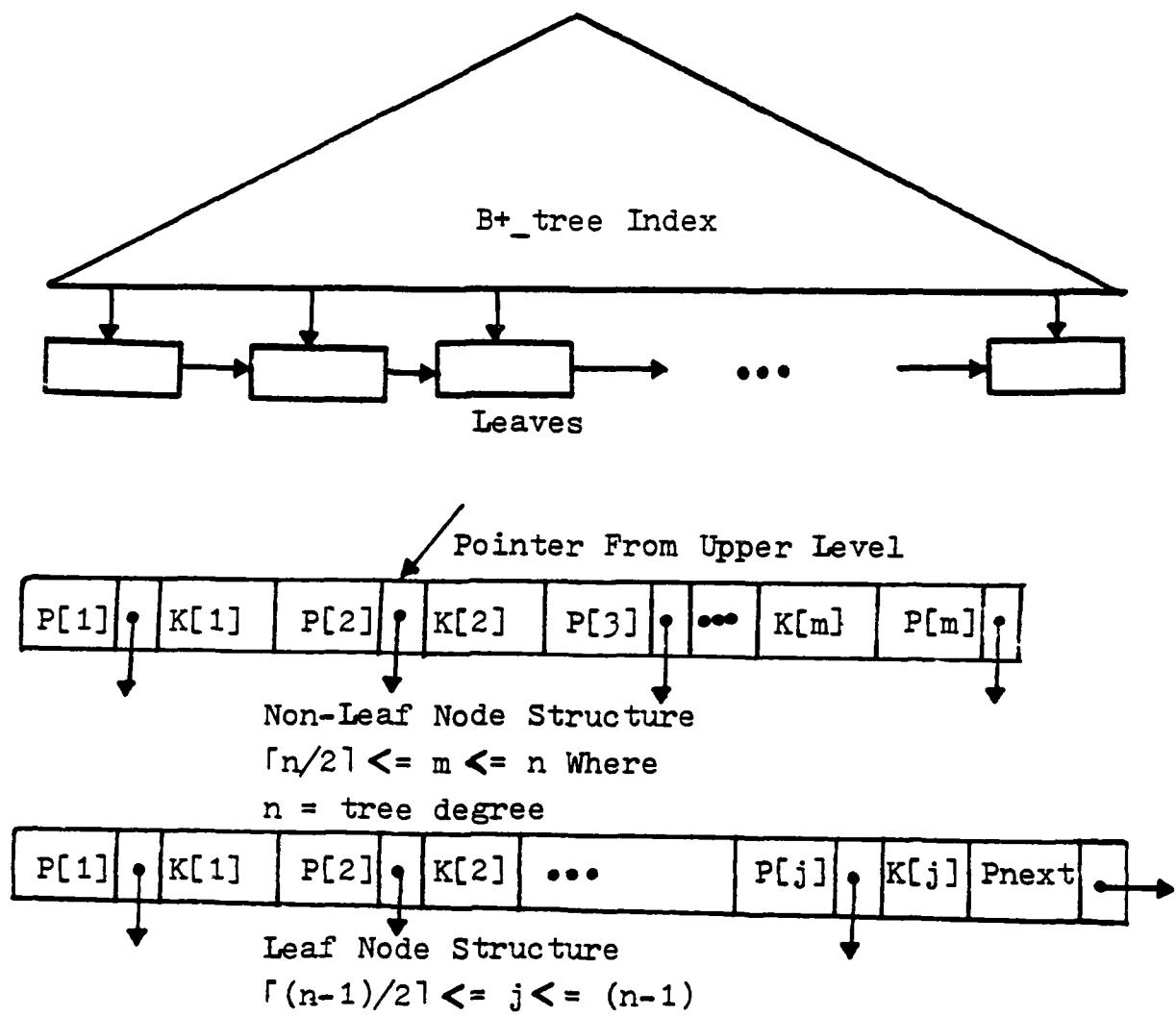


Figure 36. B+_Tree Structure

The index portion of the B+_tree is a balanced tree in which every path from the root to a leaf is the same length. As shown in Figure 36 each interior node, in this multi-level index, may hold up to n pointers and all must hold at least $\lceil n/2 \rceil$ pointers, for a tree with degree n . If we have a node with m pointers, and $1 \leq i \leq m$, then pointer $P[i]$ points to the subtree containing key values less than $K[i]$ and greater-than-or-equal-to $K[i-1]$. Pointer $P[m]$ points to the subtree containing those key values greater-than-or-equal-to $K[m-1]$ and pointer $P[1]$ points to the subtree with key values less than $K[1]$.

Each leaf node can hold between $\lceil (n-1)/2 \rceil$ and $(n-1)$ search key values. Every search key value has, associated with it, a pointer to the main file where records identified by the search key value are kept. The pointer $P[next]$ points to the next leaf in the sequence set.

During query processing a path is traversed in the tree from the root to some leaf node. If there are k search key values in the file the path will be no longer than $\log K$ base $\lceil n/2 \rceil$ meaning, in practice, only a few nodes need be accessed even for relatively large files. Typically, a node has the same size as a block of secondary storage (expensive in terms of access) to make n as large as possible. Therefore, depending on search key values, between 10 and 100 values can be stored in a node resulting in shallow trees even for very large files.

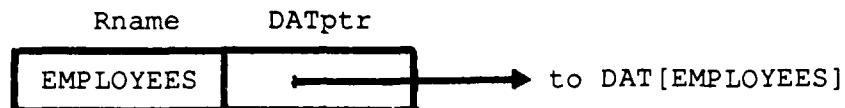
Data Access Based Upon A B+_Tree Hierarchical Index

Under this data access strategy each relation comprising the database is distributed across the RCs according to some data placement strategy. Each RC maintains a directory, for its partition of the database, consisting of some tables and the B+_tree indexes. When an RC receives a request it simply consults these directory structures to locate the desired data. The majority of the directory maintenance is accomplished independently at the RC level because each processor is responsible for its own B+_tree index structures. In this section the data structures and algorithms to support hierarchical indexing via B+_trees are presented.

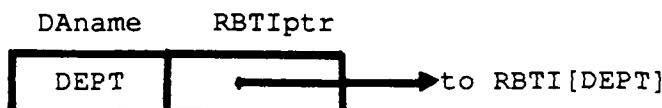
Data Structures To Support Indexed Data Access. This data access strategy requires two types of directory structure--global and local. The smaller global structures are replicated, maintained by all the RCs, and contain information about the relations and their directory attributes. The global directory structures are stored in the RC's primary memory. Local directory structures, stored in the RC's secondary memory, are the B+_trees containing the search key values and pointers to records in the relations.

A global relation information table (RIT) is required to associate each relation with its corresponding directory attribute table (DAT). Each entry in the RIT consists of the name of a relation and a pointer to the DAT (for the relation) as shown below. As before, there is only one RIT for the database,

replicated and stored in the RC primary memory.



For each relation, in the database, there is a global DAT, stored in primary memory, containing an entry for each directory attribute defined for the relation. As shown below, each DAT entry contains a directory attribute name and a pointer to the relation B+_{tree} index (RBTI) containing the key values for that directory attribute.



The final data structures, which are independently maintained by the RCs and kept in their secondary memory, are the RBTIs. Each of these structures is a B+_{tree} with pointers to records in a relation, also located in secondary memory. Each RBTI constitutes a hierarchical index, on its particular directory attribute, for a relation. The RBTIs are of the format described in Section 5.2.1, and their sizes are dependent upon such factors as the size of the key values (bytes) and number of key values stored in the trees.

An example of an RBTI, based upon the EMPLOYEES relation, is shown below.

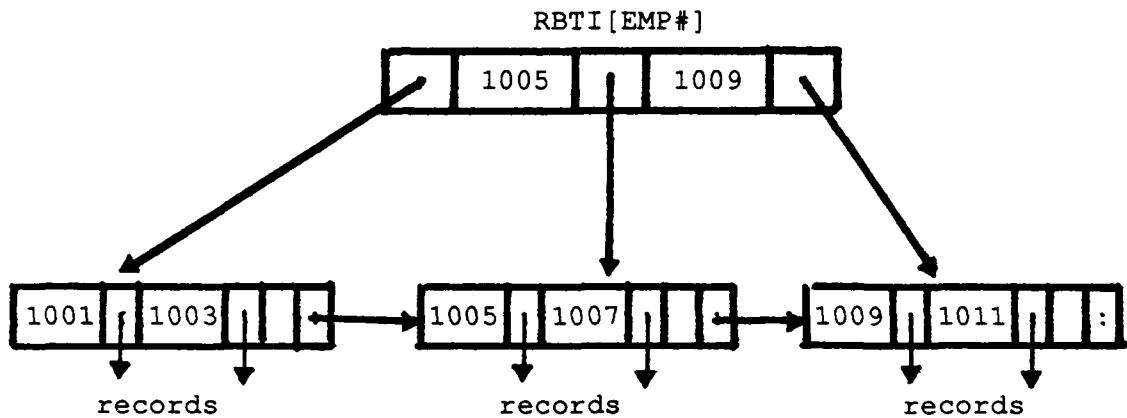


Figure 37 illustrates the directory data structures, and their relationships, for the EMPLOYEES relation example. Records of the relation have been distributed across the two-RC system by a simple round-robin data placement strategy with initial insertion at RC1. Directory attributes, as in the clustering example, are DEPT and SALARY.

Having defined the directory data structures, required to support data access under hierarchical indexing, expressions for their size may now be derived. First, a set of variables is defined forming the basis for the model. Next, the expression for the size of each structure is presented in terms of the variables. The sizes of the DAT and RBTI structures are presented in terms of smallest, largest, and the general case.

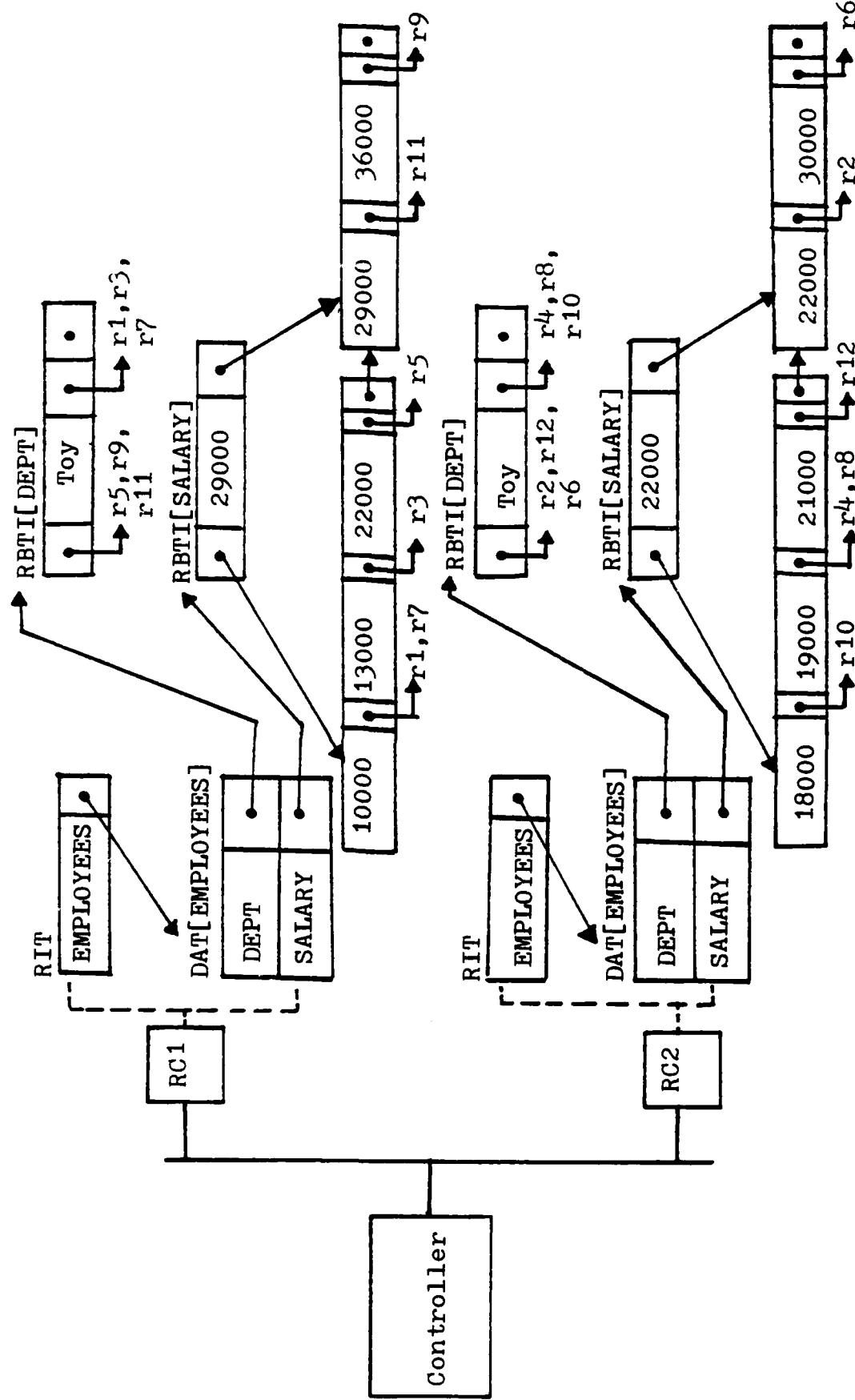


Figure 37. Hierarchical Indexing In RRDs

The following variables are used in developing the analytical model for the hierarchical index size:

<u>Variable</u>	<u>Description</u>
RC	Number of RCs in RRDS
R	Number of relations in database
Rname	Average size of a relation name (bytes)
ptr	Size of pointer (bytes)
addr	Average size of record address (bytes)
r	Number of records in a relation
a	Number of attributes for a relation
da	Number of directory attributes for a relation ($1 \leq da \leq a$)
a	Average size of attribute name (bytes)
v	Average size of a key value (bytes)
b	Disk block size (bytes)
n	Degree of an RBTI
h	Height of RBTI
nd	Number of nodes in RBTI
nl	Number of leaf nodes in RBTI
k	Number of search key values in a relation partition for a directory attribute
RIT	Size of RIT (bytes)
DAT	Size of DAT (bytes)
RBTI	Size of RBTI (bytes)

The expressions for the sizes of the directory structures under the hierarchical indexing scheme are as follows:

Structure: RIT Location: RC primary memory

Number of RITs at each RC: 1

Size:

Number entries: R
 Entry size : Rname + ptr

$$|RIT| = R(Rname + ptr)$$

Structure: DAT[R] Location: RC primary memory

Number of DATs at each RC: R

Size:

Number entries: da
 Entry size : |a| + ptr

Smallest |DAT[R]| = |a| + ptr
 Largest |DAT[R]| = a(|a| + ptr)
 General |DAT[R]| = da(|a| + ptr)

Structure: RBTI[da] Location: RC secondary memory

Number of RBTIs at each RC: da[R1]+da[R2]+...+da[Rn]
 $1 \leq n \leq R$

Size:

Degree of RBTI: $n = b/(v + ptr)$

Number leaf nodes: Smallest: $nl = \lceil k/n \rceil$
 Largest : $nl = \lceil k/(n/2) \rceil$

Height of RBTI: Smallest: $h = \log_n(k)$

Largest : $h = \log_{\lceil n/2 \rceil}(k)$

Number of nodes : $nd = \sum_{i=1}^h n^i = [(n-n^h)/(1-n)] + 1$

$$|RBTI| = \left[\sum_{i=1}^{da} nd[i] \right] (b) + (r/RC) (addr)$$

Now that the directory structures have been defined, a set of algorithms for their use in query processing, and their maintenance, is presented.

Algorithms For RRDS Hierarchical Index Directory Management. Under the hierarchical indexing data access strategy retrieval is less complex than for the clustered approach, due to the fact that records need not be accessed through clusters. The RBTIs are simply consulted to locate the records satisfying a query. Record insertion (and deletion), however, requires directory maintenance actions resulting in possibly more directory management overhead. In this section the directory management algorithms are presented, for the hierarchical indexing strategy, based upon the previously defined data structures. Index management and maintenance algorithms are provided for insertion, retrieval, and the two-relation Join operation. Each algorithm is followed by an expression for directory management time.

A record to be inserted is broadcast to the RCs. The RC which will actually perform the insertion (determined by a data placement strategy) then consults its RIT to find the directory structures for the target relation. From the RIT the DAT is accessed to determine directory attributes and locate the RBTIs

for the relation. For each directory attribute in the record the corresponding DAT entry is accessed. For each DAT entry the appropriate RBTI is located and searched for the proper leaf node for key value (value of the corresponding directory attribute) insertion. It should be noted that, if the directory attribute value in the record being inserted is already located in an RBTI leaf, no RBTI alteration is required. In the algorithm, presented here, we are considering the general case where the RBTI requires alteration. Once the proper leaf is located, the key value is inserted in accordance with the rules for B+_trees. If there is room in the leaf for the key value it is simply inserted. If the node is full then it must be split, after the key value is inserted, forming two nodes. If leaf splitting is necessary, the same procedure must be applied, to the upper level index nodes of the B+_tree, until a node is reached which requires no splitting. Details on B_trees may be found in (Comer 1979; Ullman 1982; Horowitz 1982). The insertion directory management algorithm is presented below:

Hierarchical Index Directory Management For Insert

Input: A record for insertion into a relation in the RRDS database

BEGIN

 Consult RIT to find the target relation entry.

 Get DAT for R.

 FOR each attribute field attr[1]..attr[a] in r, DO

 Consult the DAT.

 IF attr[i] is a directory attribute THEN

 Access RBTI for the directory attribute

 Search RBTI[Da] for leaf node for insertion.

 IF there are less than (n-1) keys in leaf THEN

```

        Insert key in leaf (block) in sorted order.
ELSE (* split node *)
    REPEAT
        Insert key and split the node.
        Insert a key into parent node.
    UNTIL
        No splitting necessary.
    ENDIF
ENDIF
ENDFOR
END

```

Output: The new record's key values added to the appropriate RBTIs.

The following new parameters and variables are required for formulation of an expression for record insertion directory management time:

<u>Parameter</u>	<u>Description</u>
Trit	Time to access an entry of RIT from primary memory
Tdat	Time to access an entry of DAT from primary memory
Taddr	Time to read/write a record address
Tins	Time to insert into each RBTI
Tb	Time to read/write a block of secondary memory

<u>Variable</u>	<u>Description</u>
Psp	Probability a node is full (node splitting)
RIT	Number of entries in RIT
DAT	Number of entries in DAT

The expression for time to accomplish directory management for an insert query (TDMI) is now derived.

$$TDMI = RIT/2(Trit) + a(DAT/2)(Tdat) + \sum_{i=1}^{da} Tins[i]$$

$$Tins = h(Tb) + (1-Psp)Tb + \left[\sum_{i=0}^{h-1} Psp^i (1-Psp)Tb + Psp^{i+1} (2Tb) \right] + Taddr$$

One-relation data retrieval queries are relatively simple, in terms of directory management overhead, due to the fact that indexes will not be modified. First, the RIT is accessed in order to find the directory structures for the target relations. For each predicate, in each conjunction, containing a directory attribute the DAT is consulted to find the RBTI for the predicate's directory attribute. If the predicate is an equality predicate (i.e., relational operator is "=") the RBTI is simply searched until the leaf node containing the key value is located. The good characteristics of the B+_trees are exploited for locating key values satisfying non-equality predicates. For these cases one search of the tree is required to locate an initial key value. Next, the sequence set is retrieved by traversing horizontally.

The response set for each query conjunction is the intersection of all record addresses returned for each predicate

in the conjunction. The query response is the union of the conjunction response sets. The hierarchical index directory management algorithm for retrieval operations is presented below:

Hierarchical Index Directory Management Algorithm For Retrieval Operations

Input: Data retrieval query consisting of c conjunctions
where the number of predicates in a conjunction is p.

```
BEGIN
    Consult RIT to find entry for R.
    FOR each conjunction in the query DO
        FOR each predicate in the conjunction DO
            Access DAT to find RBTI for predicate's attribute
            CASE
                Relop is "=":
                    Search RBTI[Da] to find block containing key.
                Relop is "<" or "<=":
                    Search RBTI[Da] to find leftmost block.
                    Traverse across sequence set reading blocks
                    until key value is reached.
                Relop is ">" or ">=":
                    Search RBTI[Da] to find block containing key.
                    Traverse across sequence set reading blocks.
                Relop is "<>":
                    Search RBTI[Da] for leftmost block.
                    Traverse across sequence set reading blocks.
            ENDCASE
        ENDFOR
        Compute intersection of all record addresses found for
        each predicate.
    ENDFOR
    Compute union of all record addresses found for each
    conjunction.
END
```

Output: Set of record addresses satisfying query.

The following new parameters and variables are required for formulation of an expression for hierarchical index directory management time for data retrieval:

<u>Parameter</u>	<u>Description</u>
Ti	Time to perform set operations on sets of record addresses
<u>Variable</u>	<u>Description</u>
c	Number of conjunctions in query
p	Number of predicates in a conjunction
Peq	Probability that predicate is an equality predicate
Presp	Size of predicate response set (records)

The expression for time to accomplish directory management for data retrieval (TDMS), under the hierarchical indexing data access scheme, is now presented:

$$TDMS = RIT/2(Trit) + \left[\sum_{i=1}^c Tp[i] + Ti \right]$$

$$Tp = \sum_{j=1}^p DAT/2(Tdat) + Peq(h[j](Tb)) + (1-Peq)(h[j](Tb) + (1/2)(nl)Tb) + Presp(Taddr)$$

Assumption: Sequence set traversal for { \leq , $<$, $>$, \geq } requires approximately half of the leaves. (For \leq all the leaves must be traversed, however, for generality we assumed this predicate requires half of the leaves as well.)

Directory management, under the hierarchical index strategy, for two-relation operations, such as Join, is trivial. Since operands consist of whole relations only the RIT is required. The RIT is accessed to determine the location of each relation.

Then the addresses of records comprising the relations are read directly. The time to perform directory management for a two-relation operation (TDMJ) is given below:

$$TDMJ = \sum_{i=1}^2 RIT/2(Trit) + r(Taddr)$$

Selecting A Data Access Strategy

Selection of a data access strategy, for integration into RRDS, was based upon the following criteria:

1. Directory mechanisms compatible with the replicated computer/partitioned database architecture
2. Must provide range query capability
3. Minimal response time
4. Minimal directory size

Both the clustered and hierarchical index approaches support range queries and are compatible with the RRDS architecture. The B+_tree data structure, upon which the hierarchical index scheme is based, is ideally suited to the range query environment due to the sequence sets which facilitate fast horizontal traversal. This approach also requires fewer, smaller, data structures and hence less storage space. The most important selection criterion, minimal response time, is now investigated.

The cost tradeoffs, in terms of response time, for the clustered and B+_tree schemes were compared. A simulation model, of the RC actions under each strategy, was designed and

implemented in the SLAM simulation language. The RC and secondary memory were modeled as SLAM resources. The activity times were based upon the expressions for directory management time, under each access scheme, for three query types: Insert, Select, and Join. The SLAM simulation models are presented in Appendix D. A set of experiments was designed, along with a workload model, to examine the performance of both strategies for a range of scenarios. We were specifically interested in observing the effects of database characteristics (relation size and number of directory attributes) and query characteristics (number of conjunctions, predicates, and predicate types). In addition, experiments were conducted on each strategy to observe performance for different implementations. The experimentation plan and workload models are presented in tables 10 and 11, respectively.

In addition to the assumptions discussed during directory management time derivations, the following assumptions apply to the simulation models:

1. Secondary memory access time dwarfs primary memory access time, and the size of the RIT and DAT are small relative to the size of DT, CT, and RBTI structures, therefore, access to RIT and DAT was ignored.
2. All records are fixed length.
3. All components are 100% reliable.

In addition to the variables defined in the previous sections, the following parameter values were used in both data access

TABLE 10. DATA ACCESS STRATEGY EXPERIMENTATION PLAN

Experiment 1: General Scenario

Variables	Goal
$r, da, c, p,$ Peq : Both models $d, cl, dset, pdset, Qclset,$ Pnc : Cluster model $h, k, n, l,$ $Presp$: $B^+_$ tree model Queries: Ins/Sel/Join	Observe behavior of both data access strategies for a general scenario where all variables are ranged across workload model values
Experiment 2: Database Characteristics	
Variables	Goal
r, da : Both models Queries: Ins/Sel/Join	Observe effect on response time of increasing size of database and number of directory attributes
Experiment 3: Query Characteristics	
Variables	Goal
p, Peq : Both models Queries: Sel	Observe effect on response time of increasing number of predicates and altering types of predicates
Experiment 4: Cluster Strategy Characteristics	
Variables	Goal
cl, d : Cluster model Queries: Ins/Sel/Join	Observe effect of number of clusters and number of descriptors on response time
Experiment 5: $B^+_$ Tree Strategy Characteristics	
Variables	Goal
n, r : $B^+_$ tree model Queries: Ins/Sel/Join	Observe effect of degree of RBTIs on response time as well as effect of increasing database size and RBTI degree

TABLE 11. DATA ACCESS STRATEGY WORKLOAD MODEL

VARIABLE	RANGE	UNITS
r	100 - 100,000	record
a	1 - 15	attribute
da	0 - 15	attribute
c	1 - 3	conjunction
p	1 - 10	predicate
d	0 - 40	descriptor
n	10 - 100	keys/block

models:

Parameter	Value (milliseconds)
Tdt	1.51
Taddr	.28
Tcomp	.005
Tdset	.315
Tdid	.105
Tcc	.005
Tb	18.000

Results of experiments favored the B+_tree approach over clustering. The results of running 1000 Select, Join, or Insert queries, for the general scenario, are shown in Table 12. These results represent an average response time for each of the strategies over the range of values for the variables. The results of running Select, Join, and Insert queries on a database with relation partitions ranging from 100 to 100,000 records are shown in Figure 38. For this experiment Select queries contained one conjunction with three predicates. Peq was set at .25 and the number of directory attributes was five. For the B+_tree

model, n was held constant at 20. Both strategies proved sensitive to the size of relations being accessed, however, the B+_tree approach performed consistently better and showed less degradation as the workload was increased.

The clustered scheme was especially sensitive to the query characteristics, exhibiting severe performance degradation for Select queries as the number of predicates was increased. Queries were run on a database where the size of the relation partition was 10,000 records, with five directory attributes. The number of predicates per query was varied from 1 to 10 and response time observed. The rapid degradation of the cluster model, due to calculation of the query cluster sets, is illustrated in Figure 39. The clustered scheme showed the most improvement as the percentage of equality predicates was increased. Table 13 shows the impact of the predicate type on each directory management strategy. As the probability for equality predicates, in Select queries, was varied from 0 - 1.00

TABLE 12. AVERAGE RESPONSE TIME FOR ALTERNATIVE DATA ACCESS STRATEGIES

DATA ACCESS SCHEME

QUERY TYPE	Clustered	B+_Tree
Select	48.01	40.16
Join	26.82	26.79
Insert	7.75	.09

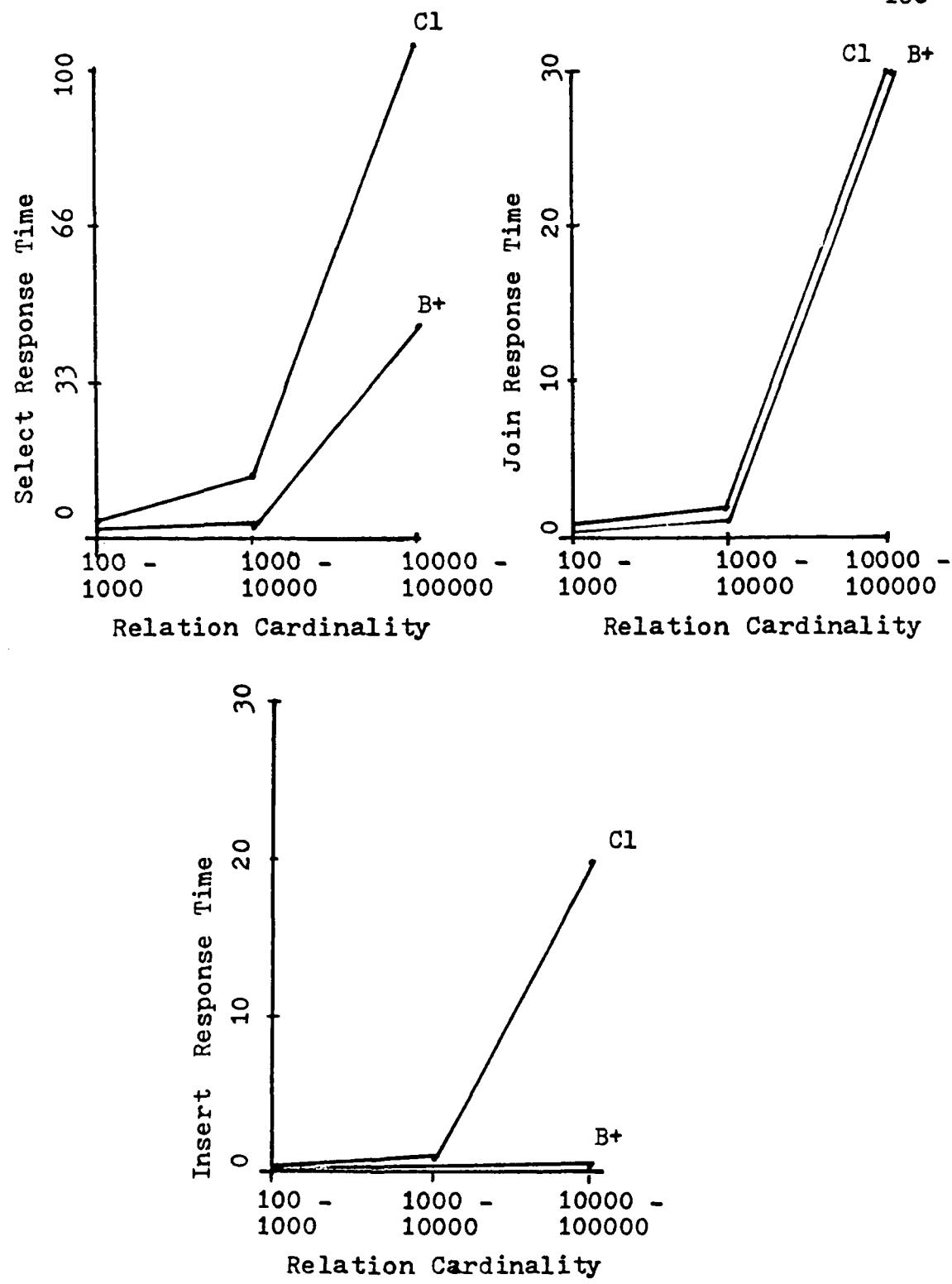


Figure 38. Effect Of Changing Relation Size

the performance of both schemes improved, however, the improvement for the clustered scheme was more pronounced. This is due to the fact that less of the DT must be searched for an equality predicate, since only one descriptor must be located for this case.

The number of clusters per relation had a profound effect upon the response time for the clustered database. For all three query types a small number of clusters per relation (i.e., between 1 and 100) gave the best response times. A relation stored as one cluster, containing all the records (i.e., no clustering), performed better than scenarios with relations partitioned into more than 100 clusters. This is due to the fact that as the number of clusters increases the overhead of searching the CT overwhelms the benefits of the partitioning effect. Worst performance, as shown in Table 14, was observed when the relation was partitioned such that each cluster consisted of only one record.

TABLE 13. EFFECT OF PREDICATE TYPES

STRATEGY	P _{EQ}		
	0 - .33	.34 - .66	.67 - 1.00
Clustered	31.81	9.24	1.95
B+ tree	9.11	5.39	1.68

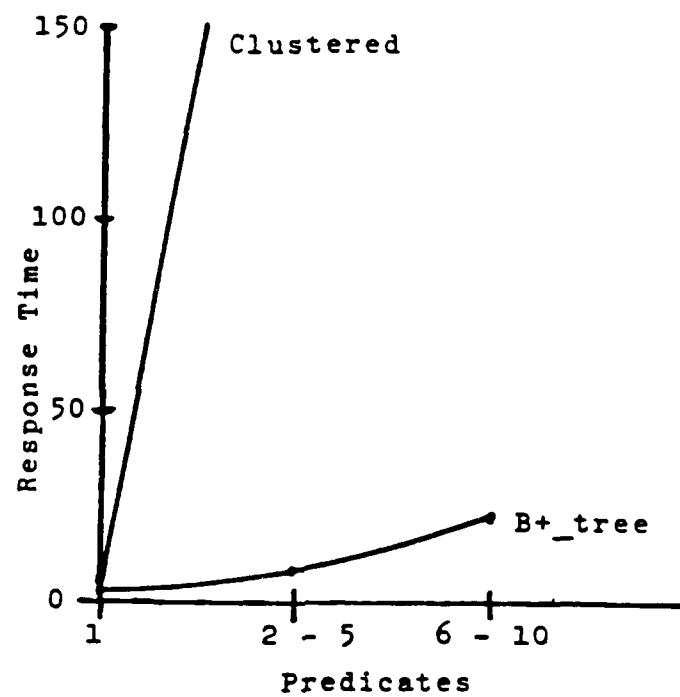


Figure 39. Increased Number Of Predicates - Response Time

TABLE 14. RESPONSE TIME FOR CLUSTERING SCHEME FOR VARIABLE CLUSTERS/RELATION

QUERY TYPE	NUMBER OF CLUSTERS				
	1	1 - 100	100 - 1000	1000 - 10000	10000
Select	2.91	.64	1.99	25.54	50.45
Join	5.60	5.60	5.60	5.60	5.60
Insert	.09	.02	.11	2.01	3.51

The results of varying the degree of the B+_trees are shown in Table 15. Queries of all three types were run against a relation of 10,000 records with five directory attributes and the degree of the B+_trees varied from 10 to 100 key values per block. As the degree increased performance improved, but not dramatically, indicating that the B+_tree approach will not degrade significantly if key values are large (i.e., long strings).

Table 16 shows the effect of increasing the size of the database and the degree of the B+_trees. Select queries were run

TABLE 15. EFFECT ON RESPONSE TIME OF INCREASING B+_TREE DEGREE

QUERY TYPE	B+_TREE DEGREE		
	10 - 40	41 - 70	71 - 100
Select	7.75	4.92	4.20
Join	5.43	5.43	5.43
Insert	.27	.23	.19

TABLE 16. RESPONSE TIME FOR SELECT QUERY AS DEGREE OF B+_TREE INCREASES AND DATABASE SIZE INCREASES

RELATION SIZE	B+_TREE DEGREE		
	10 - 40	41 - 70	71 - 100
10,000	7.75	4.92	4.20
100,000	78.90	32.84	28.05
1,000,000	768.43	486.19	417.62

against databases where the relation ranged from 10,000 to 1,000,000 records. As the degree of the B+_trees increased performance improved, indicating the benefits of the B+_tree approach for large database applications.

Based on simulation results, and due to the fact that software-oriented multi-computer relational database systems are designed for large database applications and must provide efficient range query capability, the B+_tree option should be the method of data access. This simple approach eliminates the overhead of cluster management, yielding better performance. In addition the B+_tree index can be partitioned across the processing elements saving secondary storage space.

The last results, given in Table 17, depict the relative sizes (in bytes) of the database directory, under the B+_tree scheme, for variable relation cardinalities and a variable number of directory attributes. For these calculations, record size was assumed to be a fixed-length 100 bytes. It was also assumed that

TABLE 17. DIRECTORY SIZE IN BYTES FOR B+_TREE DATA ACCESS

RELATION CARDINALITY	NUMBER OF DIRECTORY ATTRIBUTES		
	1	3	5
1,000	22,420	67,260	112,100
10,000	229,320	287,960	1,146,600
100,000	2,278,352	6,835,056	11,391,760
1,000,000	22,785,568	68,356,704	113,927,840

each B+_tree contained keys, with an average length of 20 bytes, for 75 percent of a relation's records (i.e., 25 percent of records contain duplicate key values). Block size was 512 bytes, pointers required three bytes of storage, and addresses required 6 bytes of storage. The database directory size is profoundly affected by the number of directory attributes as shown in Table 17. For the one directory attribute scenario the size of the total directory was about 25 percent that of the database, giving a directory/database ratio of approximately 4:1. As the number of directory attributes increased to three the ratio dropped to approximately 1.5:1. When the number of directory attributes was increased to five, the directory size actually exceeded that of the database resulting in a directory/database ratio of approximately .89:1.

CHAPTER 6

DATA PLACEMENT IN RRDS

In a multi-computer database architecture, such as RRDS, one of the major software design questions deals with distribution of data across the system. Will the relations, comprising the database, be replicated, partitioned, or a combination of the two? If relations are replicated, will data storage requirements become prohibitive, and if we elect to partition the database, what partitioning scheme will yield optimal performance with minimal storage requirements? These questions are addressed in this chapter. Various data placement strategies will be discussed and analyzed, with respect to the RRDS design goals, and the best scheme will be selected for integration into the RRDS design.

Different Approaches To Data Placement

Due to the fact that RRDS is intended for management of large databases data replication was not considered a viable data placement strategy. Under a replicated approach data volume would become overwhelming and performance would suffer due to the time required to maintain the replicas.

A partitioned data placement strategy must be selected which maximizes parallelism and eliminates the specialized backend problem. The goal is to have all RCs participate equally in

execution of each query. Three partitioning approaches--arbitrary data placement, round robin (RR) data placement, and value range partitioning (VRP) data placement were considered for RRDS.

Arbitrary Data Placement

Under the arbitrary strategy insertion of a record is accomplished at an arbitrary RC. This is the simplest approach, requiring virtually no data placement overhead. One of the basic criteria for partitioned data placement is that the relations be evenly dispersed across the system. The arbitrary data placement strategy does not guarantee even distribution and can result in backend limitation. In addition, one cannot predict the degree of parallelism for query execution, when records are inserted arbitrarily. Figure 40 illustrates some possible RRDS configurations under an arbitrary data placement strategy, for a 12-record relation distributed across a 3-RC system. As shown in the figure placement can range from completely lopsided to even distribution, however, in all these cases, no predictions can be made about equal participation of RCs in query processing. This approach does not provide the sophistication required by RRDS and will not be considered further. In the remainder of this chapter two alternative approaches, round robin and value range partitioning, will be examined. The relative merits of each strategy will be considered as well as the cost, in terms of performance during insertion and subsequent retrieval operations, and the best strategy will be chosen for RRDS.

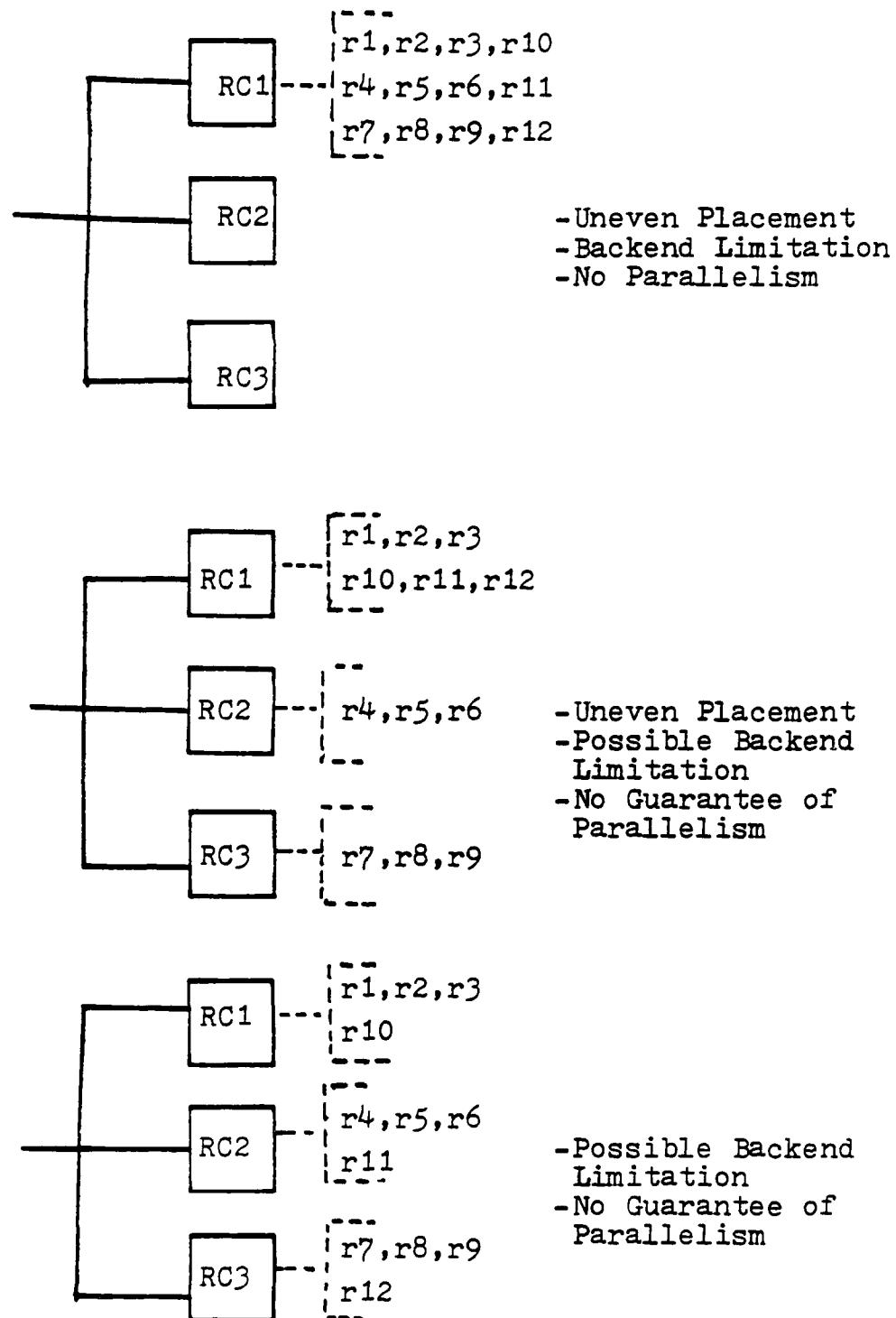


Figure 40. Possible Outcomes Of Arbitrary Data Placement

Round Robin (RR) Data Placement

A slightly more complicated approach utilizes round robin placement of records. Under such a scheme the first record of a relation is placed at an arbitrary RC. Henceforth, successive records of the relation are inserted at the RCs in round-robin order. The RR data placement scheme eliminates one of the major drawbacks of the arbitrary approach by guaranteeing even distribution of the relation across the RC network. The fact that the relations are split into equal partitions may increase the possibility of parallelism in query processing, however, in the absence of information about response set composition, the degree of parallelism (i.e., number of RCs participating) for a given query cannot be predicted. Figure 41 illustrates placement of records, of a 12-record relation on a 3-RC RRDS, under the RR placement strategy. In this example RC2 was arbitrarily chosen for insertion of the first record. Subsequent record placement followed the pattern: r2 in RC3, r3 in RC1, r4 in RC2, etc..

The main advantages of the RR approach are simplicity and low insertion overhead. The only data structures required to support this scheme are counters (nextRC), maintained at each RC for the relations, which keep track of the next RC in the RR sequence. The algorithm for RR data placement is given below:

INPUT: A stream of records for a relation R.

BEGIN

An arbitrary RC is selected for insertion
of first record of R - initialize nextRC(R).

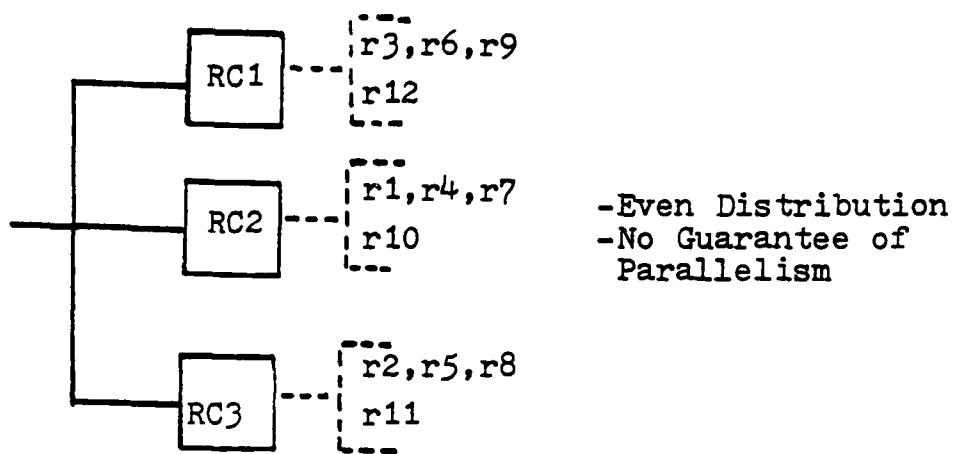


Figure 41. Example Of Round Robin Data Placement

```
FOR each record to be inserted in R DO
    Controller broadcasts record to RCs.
    B+_tree insertion is performed at nextRC(R).
    The record is inserted at nextRC(R).
    Update nextRC(R) at all RCs (i.e., nextRC(R) =
    next RC in round robin sequence).
ENDFOR
END
```

The simplicity and low cost of this approach make it an attractive choice for RRDS and, in the absence of knowledge of response set composition and user query habits, probably the best choice. This approach, however, cannot guarantee parallelism for a given query. For the situation depicted in Figure 41 a Select query with a response set of {r1, r2, r3} would be processed in parallel, however, a Select query with a response set of {r3, r6, r9} would not receive the benefit of any parallel processing. If possible, we would like to improve the probability of parallelism during query processing. The VRP data placement strategy, discussed next, accomplishes this at the cost of increased complexity.

Value Range Partitioning (VRP) Data Placement

The VRP approach, similar to the clustering technique introduced in Chapter 5, partitions each relation into evenly distributed fragments and guarantees query processing parallelism. The key to the success of this strategy is knowledge, on the part of the database creator, about user query habits and likely response set composition. VRP, however, can perform no worse than the previously discussed RR policy. The

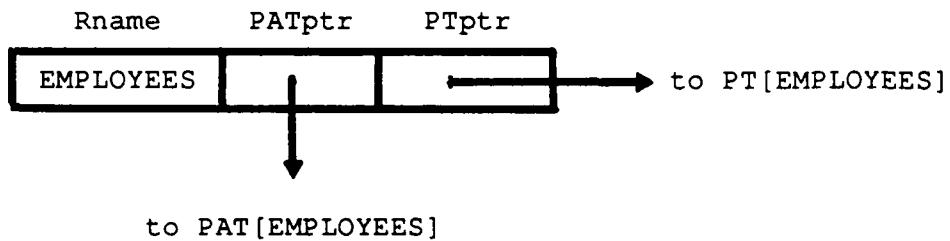
VRP strategy benefits from the ability of the clustering algorithm to partition a relation into likely response sets. A modified version of the cluster creation and insertion algorithm, discussed in Chapter 5, divides each relation into partitions according to a set of descriptors specified by the database creator. Each partition is then distributed, round-robin, across the RC network. It should be noted that the VRP strategy is designed only for data placement. Once an RC is determined for a record insertion the B+_tree insertion scheme described in Chapter 5 is used to perform the actual record insertion, and subsequent data access is via the B+_tree based directory hierarchy. Hence, this data placement strategy benefits from the parallelism derived from dividing each relation into likely response sets and distributing them across parallel processors. However, the overhead of cluster reconstruction during retrieval operations, so detrimental to the clustering data access strategy, is avoided.

The overhead incurred under a VRP strategy is in the form of more complex data structures to support data placement, and in processing required for their maintenance. The partition information must be replicated at each RC and, like the cluster information previously described, are stored in both the primary and secondary memory of the RCs. The size of the structures will be relatively small, compared to the analogous cluster directory structures, because no record address information is stored. In

the description below the EMPLOYEES relation of Chapter 5 is referenced.

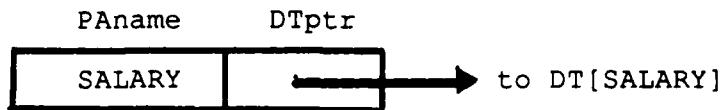
A relation information table (RIT) is required to associate each relation with its corresponding partitioning attribute table (PAT) and partition table (PT). For each partitioning attribute there is a list of descriptors maintained in a descriptor table (DT).

The RIT contains an entry for each relation in the database. Each RIT entry consists of a relation name and a pair of pointers as shown below:

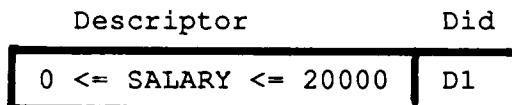


The first pointer provides access to the PAT for the relation and the second provides access to the relation's PT. There is one RIT for the database and, because of its small size, it is stored in primary memory.

For each relation in the database there is a PAT stored in primary memory. The PAT contains an entry for each partitioning attribute for the relation. Each entry consists of a partitioning attribute name and a pointer to the DT for the partitioning attribute. An example PAT entry is shown below:



For each partitioning attribute the database creator defines a set of descriptors, to partition the relation into likely response sets. The size of the DT for a partitioning attribute depends upon the number of descriptors established, for the attribute by the database creator, and can range from no entries to an entry for each value of the attribute. Each entry of the DT contains a descriptor and a descriptor identifier (Did), for example:



The partitions comprising each relation are described by the PT structures. There is a PT, in secondary memory, for each relation in the database. The PT contains an entry for each partition in the relation where an entry consists of a partition identifier (Pid), a partition descriptor set (Dset), and an RC identifier (RCid). The RCid is the identifier of the next RC for placement of a partition record. Originally, an arbitrary RCid is selected for placement of the first record. Subsequent partition records are placed in round-robin fashion. An example PT entry is illustrated below:

Pid	Dset	RCid
P1	{D1}	RC#1

Figure 42 illustrates an example of the relation partitioning data structures for the EMPLOYEES relation of Chapter 5. In this example the database creator knows that users will be retrieving records of employees with salaries less than 20,000 and records of employees who earn 20,000 or more. The relation is therefore partitioned on the attribute SALARY according to the descriptors in the DT[SALARY]. The records falling into each partition as well as the distribution of the partitions across the two RCs are illustrated in Figure 43. Note that the arbitrary RCs chosen for placement of the initial record of the two partitions are RC#1 and RC#2, respectively (see the PT[EMPLOYEES] in Figure 42). The distribution of records (in each partition) across the two RCs is even and, as illustrated in Figure 43, queries retrieving the anticipated response sets will be processed in parallel by the two RCs.

The algorithm for VRP data placement is given below. As in the cluster model, "a" is the number of attributes in the relation scheme and Rdset is a set of descriptors constructed by the algorithm for each record.

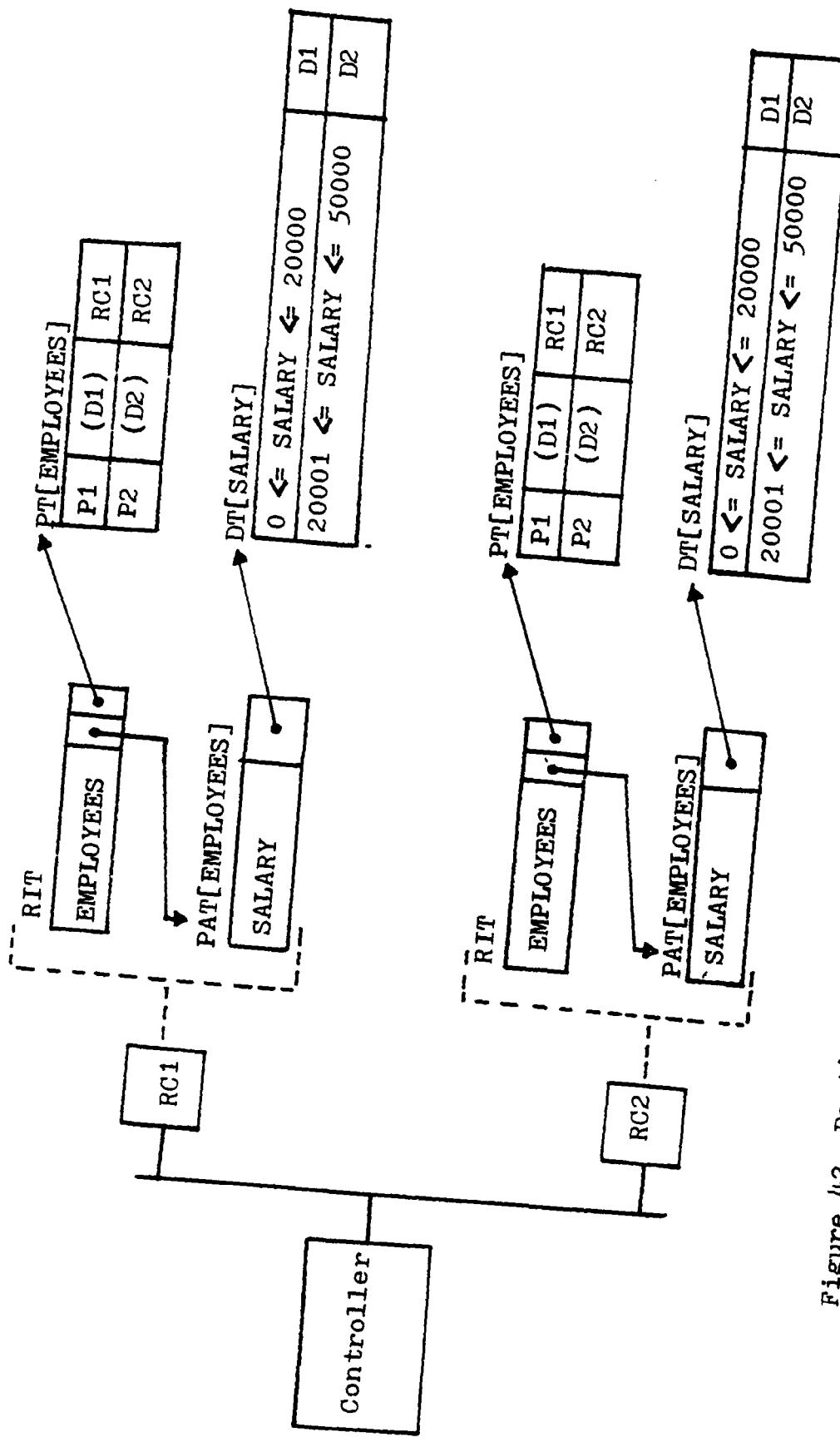
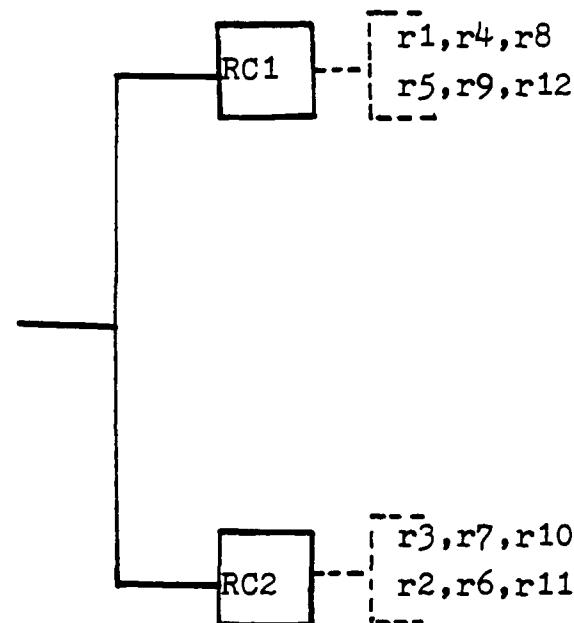


Figure 42. Partitioning Data Structures For EMPLOYEES Relation

Partition 1: r1,r3,r4,r7,r8,r10

Partition 2: r2,r5,r6,r9,r11,r12



Query: `SELECT From EMPLOYEES`

`WHERE`

`(SALARY < 20000)`

Result: `r1,r4,r8` from RC1

`r3,r7,r10` from RC2

Figure 43. Partitioning Of EMPLOYEES Relation Resulting From VRP Of Figure 42 And Assignment Of Partitions To RCs

VRP Data Placement (Record Insertion) Algorithm

Input: A record for insertion into a relation in the RRDS database.

```
BEGIN
    Consult RIT to find target relation entry.
    FOR each record, r, to be inserted in relation R DO
        For each attribute field attr[1]..attr[a] in r DO
            Consult the PAT.
            IF attr[i] is a partitioning attribute THEN
                Consult the DT.
                For descriptors in DT, until a match is found, DO
                    IF the value v[i] of attr[i] is derived
                    from this descriptor THEN
                        Add this Did to the Rdset.
                    ENDIF
                ENDFOR
            ENDIF
        ENDFOR
    Consult the PT.
    FOR each partition in the PT DO
        IF record descriptor set is equal to partition
        descriptor set THEN
            Perform B+_tree insertion at RC designated
            by RCid.
            Increment RCid.
        ENDIF
    ENDFOR
    If record not in existing partition THEN
        Create a new partition with Dset = Rdset.
        Initialize RCid to arbitrary RC.
        Perform B+_tree insertion at RC designated by
        RCid.
        Increment RCid.
    ENDIF
ENDFOR
END
```

The following new parameters and variables are required for formulation of an expression for VRP data placement time.

<u>Parameter</u>	<u>Description</u>
Trit	Time to access an entry of RIT from primary memory
Tpat	Time to access an entry of PAT from primary memory
Tdt	Time to access an entry of DT from secondary memory
Tdid	Time to read a descriptor identifier
Turc	Time to update RCid
Trrc	Time to read RCid
<u>Variable</u>	<u>Description</u>
RIT	Number of entries in RIT
PAT	Number of entries in a relation's PAT
DT	Number of entries in a partitioning attribute's DT
PT	Number of entries in a relation's PT
a	Number of attributes for a relation
pa	Number of partitioning attributes for a relation ($0 \leq pa \leq a$)
Ppa	Probability that an attribute is a partitioning attribute
dset	Size of Dset (bytes)
Pnp	Probability that a record does not belong to an already existing partition (probability of new partition)
Btree	B+ <u>tree</u> insertion time

The expression for time to accomplish VRP placement is now derived. It is assumed that, on the average, locating partitioning attributes, descriptors, and partitions will require

searching half of the respective data structures.

$$T_{VRP} = T_1 + T_2 + T_3 + T_4$$

Where:

$$T_1 = RIT/2(T_{RIT})$$

$$T_2 = a(PAT/2)(T_{PAT})$$

$$T_3 = pa(DT/2)(T_{DT})$$

$$T_4 = (1-P_{NP})\{(PT/2)[dset(T_{DID}) + T_{RRC}]\} + \\ P_{NP}\{(PT[dset(T_{DID})] + dset(T_{DID}) + T_{RRC}) + \\ B_{tree} + T_{URC}\}$$

The expression above gives the overhead incurred as a result of using VRP for data placement. Next, we are interested in determining if the parallel processing realized during data retrieval justifies imposition of this overhead.

Analysis of RR and VRP

An RRDS, operating under RR and VRP data placement strategies, was simulated and analyzed with respect to data placement overhead during record insertions and performance improvement during record retrievals. Experiments were designed to determine whether the advantage of parallelism under the VRP policy is worth the overhead of partition maintenance. The degree of overhead incurred was determined by running Insert operations on RRDS models using each of the placement strategies. VRP characteristics, such as the number of records per partition,

number of partitioning attributes, and number of descriptors per partitioning attribute were varied and response times observed.

The average response times for Insert operations, on 10,000 record relations, under both policies are shown in tables 18 and 19.

As shown in Table 18, for a small number of partitions (i.e., few partitioning attributes) the overhead of the VRP strategy was not

TABLE 18. EFFECT OF NUMBER OF PARTITIONING ATTRIBUTES ON INSERT RESPONSE TIME

DATA PLACEMENT STRATEGY	PARTITIONING ATTRIBUTES		
	NUMBER OF PARTITIONS		
	0 - 1.6 1 - 10	1.7 - 3.3 11 - 100	3.4 - 5 101 - 1000
VRP	.392	.403	.583
RR	.389	.389	.389

TABLE 19. EFFECT OF DESCRIPTORS/PARTITIONING ATTRIBUTE ON INSERT RESPONSE TIME

DATA PLACEMENT STRATEGY	DESCRIPTORS PER PARTITIONING ATTRIBUTE			
	NUMBER OF PARTITIONS			
	1 - 3 1 - 27	4 - 6 64 - 216	7 - 9 343 - 729	10 - 12 1000 - 1728
VRP	.395	.423	.530	.834
RR	.389	.389	.389	.389

significant relative to the RR policy. However, as the number of partitioning attributes and resulting partitions increased, the cost of their maintenance, under the VRP strategy, became prohibitive. Similarly, as illustrated in Table 19, few descriptors per partitioning attribute had little adverse effect on response time of VRP. However, as the number of descriptors increased, the number of partitions increased, resulting in degraded performance. Although both the number of partitioning attributes and number of descriptors per attribute affect the number of partitions, and hence the VRP performance, increased numbers of partitioning attributes had the most impact.

Select queries were run on two models of RRDS designed to simulate the degree of parallelism under both data placement strategies. Figure 44 shows the impact of degree of parallelism and query response set size on performance. The size of response sets, for relations of 50,000 records, was varied from 1 to 50,000 records and degree of parallelism for the RR policy varied from 1 to n , where n is number of RCs in the system. Simulation runs of Select queries revealed that for small response sets (1 to 1,000 records) the ratio RR/VRP was small (approximately 1.01). However, as the size of the response sets increased so did the benefit of VRP parallelism. For large response sets (10,000 to 50,000 records) the ratio RR/VRP was 1.51, a significant improvement in performance. Finally, Figure 45 illustrates the performance of different RRDS configurations under both data placement strategies, for Select queries with

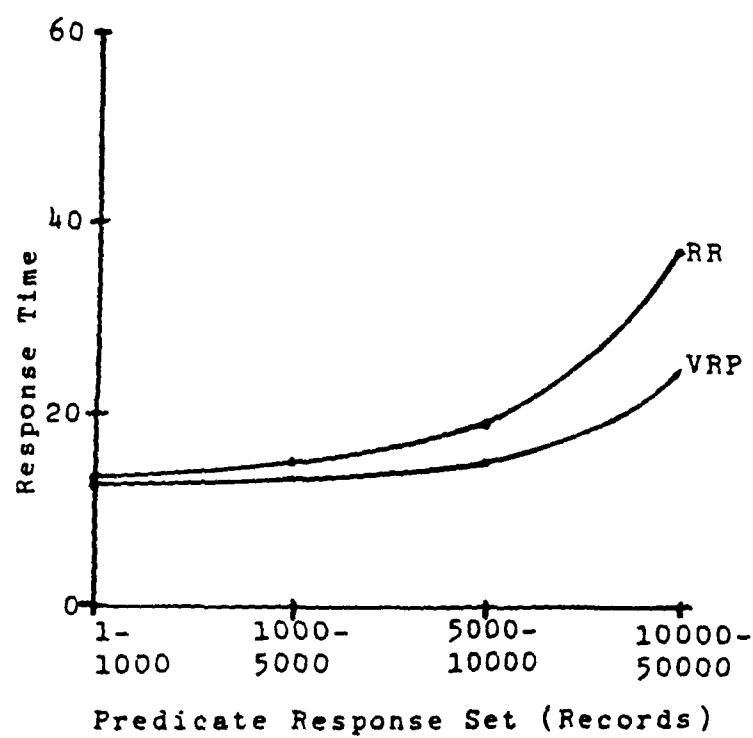


Figure 44. Effect Of Degree Of Parallelism And Response Set Size On Performance

large response sets. As the number of RCs comprising the system increases so does the performance benefit derived from a VRP data placement strategy. However, as the number of RCs increases, the performance, under RR, fails to improve proportionally.

A Data Placement Strategy For RRDS

In RR, the simplest strategy, memory requirements are minimal. The first record of a relation is placed at an arbitrary RC. Successive records are inserted round robin and RCs need only maintain the identifier of the next RC for insertion. The advantages of this approach are simplicity and the fact that the records of each relation are distributed evenly across the system. This even distribution does not, however, guarantee parallelism in query processing. Under this policy there is no way to guarantee a query will access records on all the RCs and not a subset of the RCs. The VRP policy, which requires more memory and processing overhead, guarantees that queries will be processed equally on all the RCs. In this scheme the relation is divided into value range partitions based upon descriptors. Each partition is then spread evenly across the system. Under VRP, if partitions are created based upon query characteristics, then all the RCs will participate equally in query execution.

Since RRDS is targeted for large database applications, and improved response time is more desirable than memory savings, the VRP option should be the data placement strategy of choice for

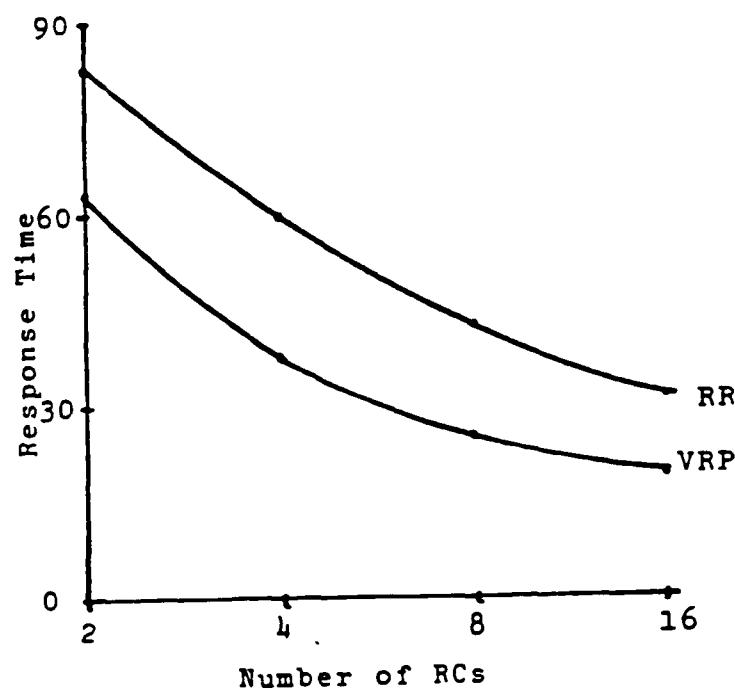


Figure 45. Effect Of Increasing Number Of RCs

large relations. If relations are partitioned, into a few relatively large partitions, the overhead incurred during Insert operations is acceptable given the fact that complete parallelism is guaranteed, for query processing, under the VRP strategy. In the case of small relation sizes the RR placement strategy would be sufficient. RRDS will accommodate both approaches to data placement, and the decision of whether or not to partition a relation must be made by the database creator, based on the relation size and user query habits.

CHAPTER 7

DIRECTORY MANAGEMENT IN RRDS

In the previous chapters data access and placement schemes were developed for RRDS. The characteristics of the directory structures and their sizes, as well as the database partitioning scheme, have been determined. The next question to be answered is how, and where, will the directory itself be maintained? We can expect the directory, especially for very large databases, to be large requiring significant processing time. The directory management should, therefore, be parallelized as much as possible. In addition, the original design goals of RRDS must be remembered--minimal controller limitation, no specialized backends, and minimal communication overhead.

Directory management is the entire sequence of actions, taken by RRDS, from the time a correct query is received to the time the secondary storage addresses of the target records are generated. It consists of accessing and maintaining the directory data structures, described in Chapter 5, using the described algorithms. In this chapter we explore directory management strategies for multiple backends in terms of directory size and performance. Previously, it was assumed the directory would be partitioned across the RC network and processed in parallel during each query. Intuitively, this still seems to be

the best approach, however, other strategies must be explored.

Alternative Strategies Under Consideration

Five directory management strategies were considered for RRDS, ranging from completely centralized approaches to schemes where RCs manage their own directories independently. Each approach, with its inherent advantages and disadvantages, is presented below.

Controller Directory Management

Under this strategy the entire directory is located at, and maintained by, the RRDS controller. The main advantage of this completely centralized approach is simplicity, however, the disadvantages eclipse this benefit. First, controller directory management violates the fundamental design goal of minimizing controller limitation. A system controller managing the entire directory creates a severe bottleneck at that processor, defeating the advantage of having multiple backend processors. The fact that the RC location for each record needs to be maintained only exacerbates the problems already mentioned. In addition, communication overhead increases as record locations are broadcast to the RCs along with queries. The controller limitation and increased message overhead inherent in this strategy are contrary to the design goals of RRDS, therefore, controller directory management is eliminated from contention for RRDS implementation.

Dedicated RC Directory Management

In order to avoid controller limitation, the next logical step is to explore the possibility of moving the directory management function to the RC network. In centralized single RC directory management the entire directory is maintained by a designated RC. As before the advantage is simplicity. However, the major disadvantage of the previous approach still exists. Instead of controller limitation, the bottleneck now occurs at the designated RC since it is required for directory management in every query. In the S-Arch experimentation of Chapter 4 the drawbacks of this approach were observed. Directory management by a "special backend" caused a severe bottleneck to occur at that backend, resulting in backend specialization and limitation. The problem of increased communication overhead also exists here since RC locations must be broadcast with queries. This strategy is also dismissed from further consideration due to the increased communication overhead and specialized backend limitations that exist.

Rotating Directory Management

Instead of a single centralized directory at a special processor, this strategy replicates the entire directory at all the processors, including the controller. Directory management is performed round-robin on the processors. Specifically, directory management for the first query is performed at the controller, the second query at RC1, the third query at RC2 and

so forth. This approach may relieve the controller somewhat from the burden imposed under the first strategy (controller directory management), however, controller limitation will still occur to a significant extent. This strategy also has the problem of maintaining a complete directory at each processor. Furthermore, there is increased message traffic between the processors resulting in increased communication overhead. Since a fundamental goal of RRDS is to eliminate, as much as possible, controller limitation this approach to directory management is also eliminated from further consideration.

Rotating Directory Management Without Controller

The controller bottleneck created in the previous strategy can be avoided by removing the controller from the round-robin sequence. Now, each RC has a copy of the directory and they take turns performing directory management for the queries. In this strategy, record addresses for different queries can be generated by different RCs in parallel, i.e., RCs can perform directory management for different queries in parallel. Since the controller limitation problem does not exist in rotating directory management without controller (RWOC), this strategy will be considered for further evaluation.

Partitioned and Parallel Processed Directory Management

In this approach, the directory is partitioned across the RC network. More specifically, the B+tree data structures at each RC index only the records stored at that particular RC. Each RC

performs directory management on its portion of the directory for each query. Instead of having all of the directory processing for a particular query accomplished at a single RC, as in RWOC, directory management for each query is now accomplished in parallel by all the RCs (operating independently on their own directory partitions). The advantages of this scheme include no controller limitation or backend specialization. In addition, the need to broadcast record addresses is eliminated because the address generation performed at a particular RC will be for the records stored at that RC. Finally, the size of the directory partition, at each RC, is inversely proportional to the number of processors in the RC network, and changes to the directory need not be accomplished at all the RCs. For these reasons the partitioned and parallel processed (PPP) approach appears to be a good candidate for RRDS. In the next section the two promising strategies, RWOC and PPP, will be compared in terms of directory size and directory management overhead.

A Comparison Of RWOC And PPP Strategies

Under RWOC, requiring full directory replication, the directory is considerably larger than for the PPP scheme. Data structures (RIT, DAT, and B+_trees) for the entire database are stored at each RC under the RWOC approach. Additionally, under RWOC, each record address contains the RC identifier (RCid) where the record is located as well as the secondary storage address (disk number, cylinder number, and track number).

The replicated directory size at each RC for RWOC directory management is given below:

$$|RIT| + \sum_{i=1}^R \left[|DAT[i]| + b \left[\sum_{j=1}^{da} nd[j] \right] + r(addr + RCid) \right]$$

Where nd is calculated as in Chapter 5, except that the variable k now represents the number of keys for the entire relation instead of for a relation partition stored at an RC. As mentioned before, a larger address, including the $RCid$, is required for each record.

Under the PPP strategy the directory size is a function of the number of RCs in the system, since the directory is partitioned. The RIT and DAT are replicated, as in RWOC, but the B+ tree structures (RBТИs) at each RC are only for the records located at the RC. Since no exchange of addresses is required, under PPP, the record address contains only the secondary storage address, and not $RCid$. Under PPP the size of a directory partition managed by an RC is as follows:

$$|RIT| + \sum_{i=1}^R \left[|DAT[i]| + b \left[\sum_{j=1}^{da} nd[j] \right] + (r/RC)(addr) \right]$$

Where nd is calculated as in Chapter 5 and RC is the number of RCs in the system. It can be seen, from these two expressions, that the directory under the RWOC scheme requires more storage

than that under PPP, and as the number of RCs increases, directory size under PPP decreases proportionally. Bearing this fact in mind, the next task is to evaluate the performance of the two strategies.

It must be determined whether processing multiple queries in parallel (RWOC) outperforms parallel processing single queries (PPP), considering the respective directory sizes. The performance criterion is query response time which is directly impacted by the queuing effect produced under each strategy. SLAM simulation models (Appendix F) of a three-RC RRDS were created for each directory management strategy, for Select and Insert queries, and experiments were conducted to observe the performance.

In the RWOC model, queries are dispatched round-robin to the three RCs which perform the directory management actions of Chapter 5, on a replicated directory, described by the workload model. For this model a 2MB/sec bus, similar to the Mass Bus of the VAX 11/780 (Baer 1980), was assumed. The new parameter Tadtrans, with a value of .004ms, was introduced to represent the time needed to transmit an address over the bus. Following generation of the record addresses for each query, in RWOC, they are broadcast via the bus to the other RCs. In the PPP model, directory management is accomplished on each RC's directory partition. Under this scheme there is no need to broadcast the addresses, once they are generated, since each RC has only its own record addresses.

A set of five experiments was conducted comparing both models subject to the workload model of Chapter 5. Experiments were designed to observe the effects of user characteristics (query interarrival times, response set sizes), database characteristics (target relation cardinality), and query characteristics (types of predicates comprising query). Results of all experiments, discussed next, favored the PPP approach to directory management.

The results of running Select and Insert queries for a general scenario are shown in Table 20. These results represent the average response time for RWOC and PPP using a range of variable values for the workload model. PPP performed, on the average, approximately three times better than RWOC for Select queries. Performance was roughly the same for Insert queries due to the minimal directory management required for such queries. The small difference in performance for Insert queries is due to

TABLE 20. AVERAGE RESPONSE TIME FOR ALTERNATIVE DIRECTORY MANAGEMENT STRATEGIES

DIRECTORY MANAGEMENT STRATEGY	QUERY TYPE	
	SELECT	INSERT
RWOC	43.31	.31
PPP	13.85	.28

the requirement to update the replicated directories, via the bus, after such operations.

Both directory management strategies were sensitive to changes in query interarrival times for Select queries. One hundred queries were run against a database with relations containing 10,000 records. Query interarrival times were assumed to be Poisson with mean varied from 1 to 50 seconds to observe queuing effect. At an average interarrival time of 1 second both systems were utilized 100 percent, resulting in long queue wait times and poor response times. As the mean interarrival time was increased queuing effect diminished, producing better response times. As the graph of Figure 46 shows, at saturation, both RWOC and PPP performed approximately the same, however, as the interarrival time increases system utilization drops (approximately 65 percent at Mean IAT = 5sec) and the PPP strategy outperforms RWOC.

The amount of information requested by users, the size of query response sets, was also varied in order to observe the performance of the alternative directory management strategies. The size of the response set impacts the number of addresses which must be broadcast, via the bus, under RWOC. Response sets were calculated as a percentage of the relation size, where relations varied from 10,000 to 100,000 records. Response sets containing .1% to 100% of the relation's records were considered for queries arriving, on the average, every 20 seconds. The results are illustrated in the graph of Figure 47. As shown, the

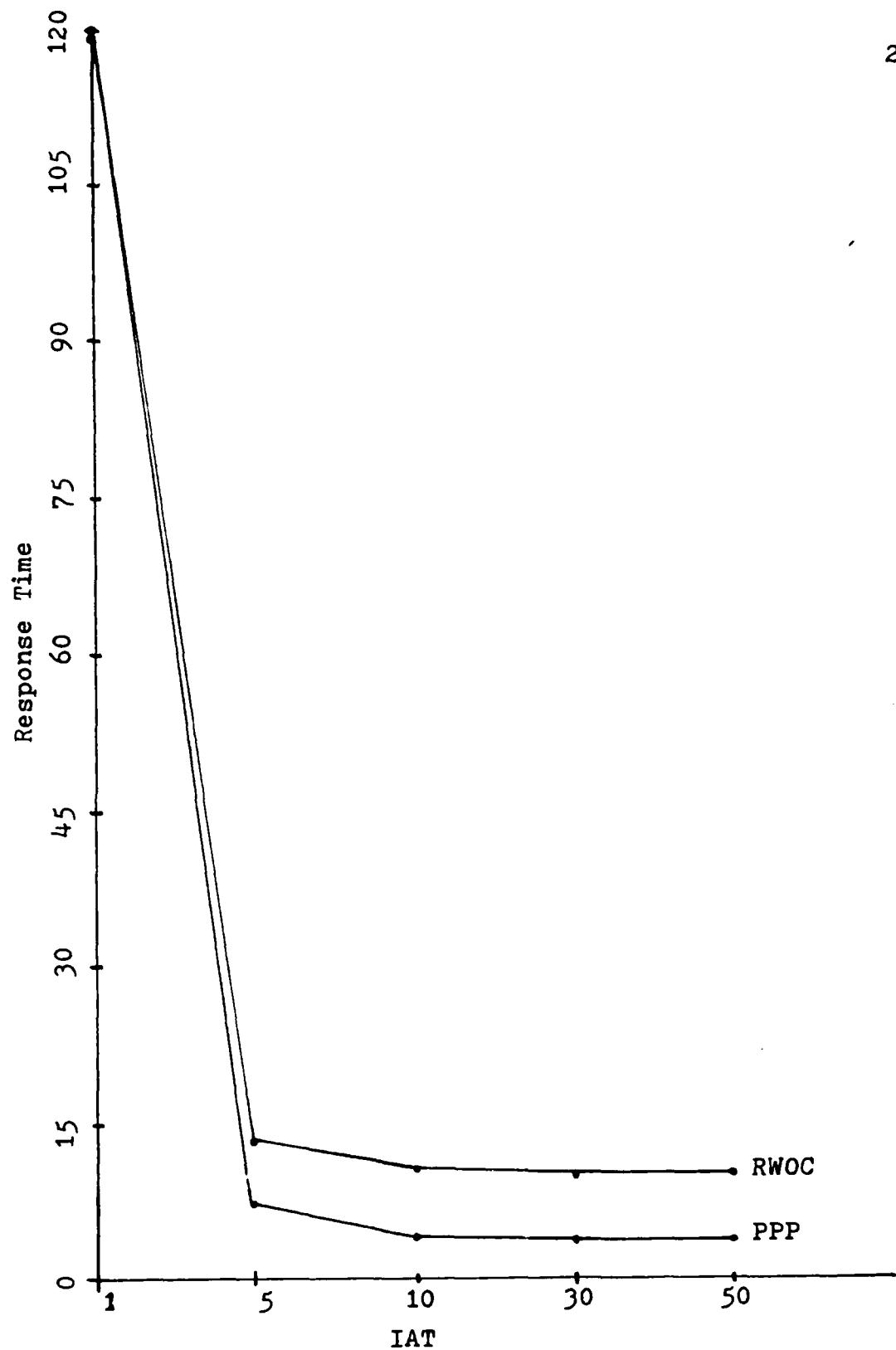
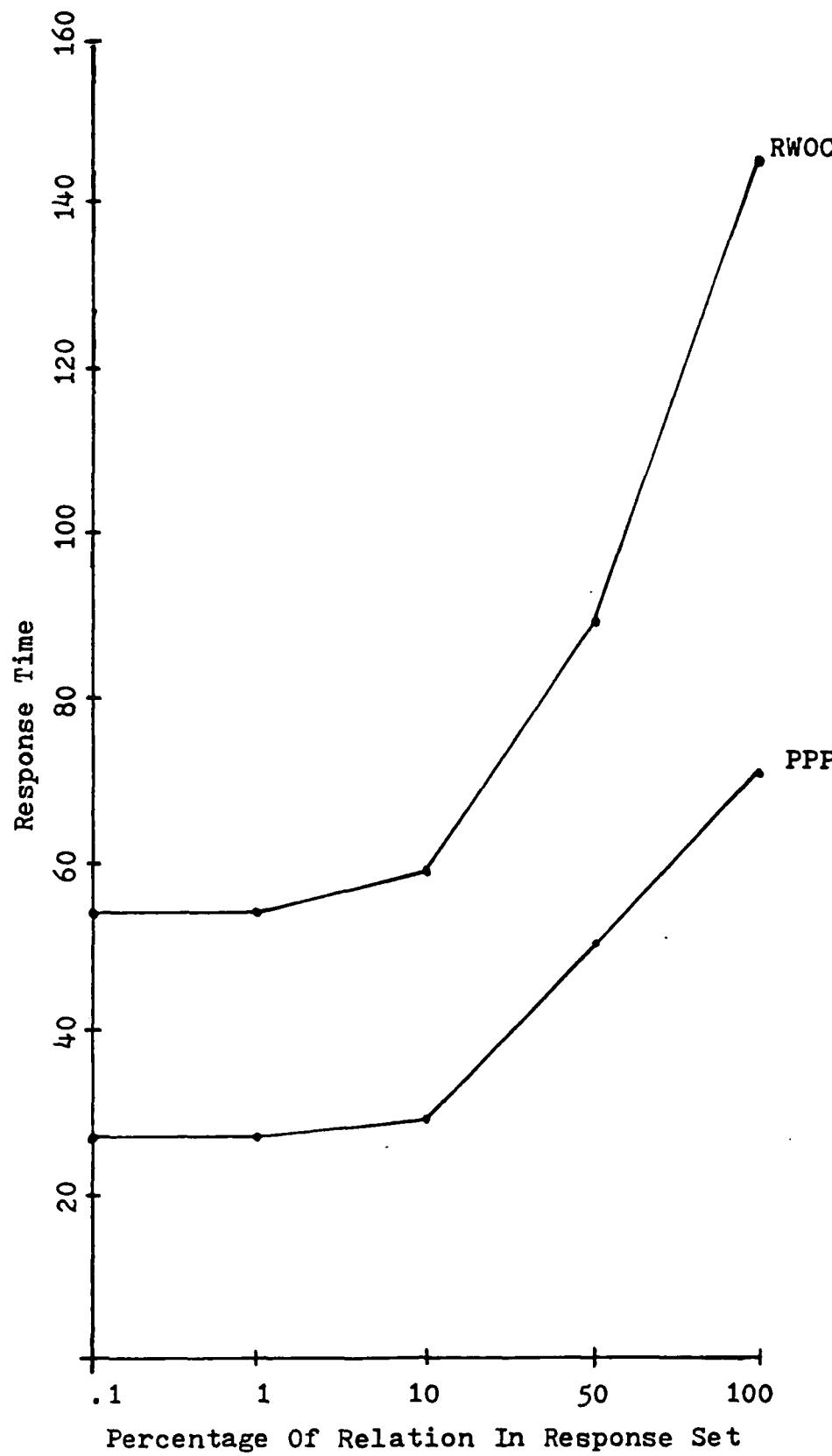


Figure 46. Response Time Versus Interarrival Time
For Select Queries



Percentage Of Relation In Response Set

Figure 47. Effect Of Response Set Size On Select Response Time

PPP strategy performed approximately 100 percent better across the entire range of response set sizes. An even more interesting observation is the fact that, as response sets increased from 50% to 100%, the RWOC performance deteriorated disproportionately to the PPP performance, indicating the overhead associated with broadcasting the addresses on the bus. The significant contributing factor to the better performance of PPP, however, is the fact that the directory is partitioned, requiring searches of smaller B+_trees and less secondary memory access.

The factor most affecting the size of the directories is the cardinality of relations in the database. The results of increasing the number of records per relation, for Select queries, are given in Table 21. Average query interarrival time was 20 seconds and the relation size was varied from 100 to 100,000 records/relation. Both strategies were sensitive to changes in relation size with the PPP approach performing approximately 100 percent better over the entire range. The

TABLE 21. RESPONSE TIMES FOR DIRECTORY MANAGEMENT STRATEGIES FOR VARIABLE RECORDS/RELATION

RECORDS PER RELATION

STRATEGY	100 - 1,000	1,000 - 10,000	10,000 - 100,000
RWOC	.634	5.49	145.02
PPP	.259	2.04	71.57

impact of directory size is highlighted by the rapid performance degradation, as relation size is increased, emphasizing the benefits of the smaller, partitioned, directories characteristic of the PPP strategy.

A final experiment examined the impact of the type of predicates in the queries. Equality predicates require less secondary memory access due to the fact that, in most cases, only one record will satisfy the query. As expected, for relation sizes ranging from 10,000 to 100,000 records and query conjunctions consisting of three predicates, when the probability of equality predicates (P_{eq}) increased, the directory management time decreased for both strategies.

TABLE 22. EFFECT OF PREDICATE TYPES ON DIFFERENT DIRECTORY MANAGEMENT STRATEGIES

STRATEGY	P_{eq}		
	0 - .33	.34 - .66	.67 - 1.0
RWOC	60.79	29.74	8.24
PPP	38.79	13.25	3.18

As shown in Table 22, PPP consistently outperformed RWOC as P_{eq} was increased from 0 to 1.0, however, degradation in performance for PPP was more pronounced than for RWOC as P_{eq} decreased.

A Directory Management Strategy For RRDS

The smaller directory structures, coupled with the reduced processing overhead, of the PPP strategy make it the choice for RRDS. The partitioned approach along with the parallel processing of each query, featured in PPP, agree with the original design goals set forth for this system. This strategy uses minimal secondary storage, creates no controller bottleneck or backend specialization problems, and reduces communication overhead (since it does not require update of replicated directories).

This concludes the design phases for RRDS. A hardware configuration has been proposed along with appropriate strategies for data access, data placement, and directory management. In the next chapter query processing in RRDS will be discussed, in detail, and the results of a comprehensive performance analysis will be presented.

CHAPTER 8

RRDS PERFORMANCE ANALYSIS

RRDS is distinguished in its application of the relational data model in concert with a database partitioning scheme for isolating response sets and refining response set granularity. In addition a partitioned, parallel processed, B+_tree directory provides efficient range query data access without the overhead of cluster management. The system has been designed with the goals of extensibility, and performance proportional to the number of processing elements, in mind.

In this chapter the results of a predictive performance analysis, based upon simulations, are evaluated relative to the original design goals. First, query processing in RRDS is described. High-level algorithms, providing the basis for the performance analysis model, for each relational operation available in the system are described. Next, the performance analysis goals, methodology, and approach are outlined. Details of the RRDS simulation model are provided including descriptions of the parametric and workload models. An experimentation plan is developed and carried out, and results analyzed with respect to the RRDS design goals. The conclusions section gives insight into how well the proposed design meets the original criteria and provides some basis for future research.

Query Processing In RRDS

RRDS query processing facilities are an implementation of the entire set of operations available under the relational algebra. In addition, aggregate capability is available, including: count, sum, min, max, and average. All queries are passed from the host to the RRDS controller where they are parsed and lexically analyzed. A proper request is then broadcast to all the RCs for further processing. The type and extent of the processing performed by the RCs depends upon the operation specified in the query as well as whether one or two relations are involved. Regardless of the query, the only portions of its processing performed serially are those performed by the controller. All other operations, such as address generation, database search, and record manipulation are accomplished in parallel by the RCs. This strategy complies with the design goal of minimizing message traffic, eliminating bottlenecks, and maximizing parallelism.

One-Relation Queries

One-relation operations in RRDS are performed on either whole relations or portions of relations. The common characteristic of the one-relation operations is that there is only one operand, either a relation or portions from one relation. In the following sections actions taken by the RCs for processing one-relation queries are described.

The Select And Project Operations. Select operations are used to obtain specified sets of records from a relation. For example, the query:

```
SELECT FROM R
  WHERE
    <specifier>
```

will cause the retrieval of the records in relation R which satisfy the specifier clause. Upon receiving a Select query each RC will consult its B+_tree directory to determine which records are affected. These records are fetched from the secondary storage and checked against the query specifier. The results, records satisfying the query, are collected into temporary buffers at each RC. Finally, the RCs transmit the contents of the temporary buffers to the controller where they are assembled into a single temporary relation and forwarded to the host.

The Project operation yields a vertical subset of a specified relation, i.e., the unique values obtained by taking specified attributes in a relation. For example, the query:

```
PROJECT <A1,A2,...,An> FROM R
```

returns the unique values assigned to the attributes A1 through An for all the records in relation R. The first step, performed by the RCs, in processing this Project query is to determine all records comprising the desired relation and their addresses in secondary memory. These records are then fetched and the values for the attributes specified in the query are stored into temporary buffers, and subsequently transmitted to the controller. Upon receiving the contents of all temporary

buffers, from the RCs, the controller assembles them into a temporary relation and eliminates duplicate tuples. The temporary relation is then forwarded to the host by the controller.

Rarely will a user be interested in a complete vertical subset of a relation. A more common query is a qualified projection, combining selection and projection operations, as shown below:

```
PROJECT <A1,...,An> FROM R
      WHERE
            <specifier>
```

A query of this type is performed exactly as a selection until the records satisfying the qualification are located by each RC. At this point, instead of collecting each record into the temporary buffers, only the values of the attributes specified in the attribute list are placed in the temporary buffers. The contents of the temporary buffers are then transmitted to the controller where duplicates are eliminated and the resulting relation is forwarded to the host.

The Insert Operation. The Insert operation is used to add a new record to the database. The command format is illustrated below:

```
INSERT <V1,V2,...,Vn> INTO R
```

V1 through Vn represent the new record to be inserted into the relation R (the schema for R has n attributes). As new records are inserted the system must ensure that the database (more specifically, each small relation or partitions of each large relation) remains evenly distributed across the RCs in order to

maximize parallelism and prevent any RC from becoming a bottleneck. This partition management function is accomplished by the RCs, reducing the controller's workload. Following the initial distribution of the records in a relation/partition across the RCs at database creation, the system maintains the even distribution of data in a round-robin fashion. All of the RCs are numbered and an RC is picked arbitrarily to insert the first record in a new relation/partition. Subsequently, all of the RCs maintain a counter indicating which RC performed the last insertion for a relation/partition, and when a new insert request arrives, the next RC in line inserts the record into its relation/partition.

As in all other RRDS operations, Insert requests are broadcast to all the RCs. When a new record arrives for insertion all the RCs perform the following processing. First, if the relation is partitioned, the attribute values specified in the record being inserted are checked against the descriptor specifications, in the directory partition data structures, to determine which partition will receive the new record. Once the proper relation/partition is determined, the next RC in line, for an insertion into that relation/partition, stores the new record into its secondary storage. This RC then sends an "Insertion Complete" message to the controller. All the other RCs discard the insert request, and increment their counters indicating the next RC to insert a record into this relation/partition.

Finally, the controller transmits the "Insert Complete" message to the host.

The Delete Operation. Records are deleted from the RRDS database by the following command:

```
DELETE FROM R
WHERE
    <specifier>
```

This query is processed exactly as a Select operation where records satisfying the qualification condition(s) are identified and located by all the RCs. Once the candidate records are located, in the secondary storage, they are marked as deleted (the records are actually removed from the database at database reorganization). Following deletion of the records a "Deletion Complete" message is sent to the controller, from the RCs, for transmission to the host.

The Update Operation. Update operations are used to change values in records in the RRDS database. These commands are performed exactly as projections except that the values for an attribute are modified instead of being retrieved. Once RC processing is complete an "Update Complete" message is sent to the controller for transmission to the host.

A more common update operation changes values in some of the records in a relation, identified by a qualification clause. When the RCs receive a qualified update, processing proceeds as in a Select operation to identify the affected records. Once the

records satisfying the qualification predicate(s) are located the update is performed exactly as described above.

Aggregate Operations

In RRDS aggregate functions are used to calculate certain statistics about groups of records in the database. Capabilities supported by this system include min, max, count, sum, and average. Aggregate operations are specified as a prefix to an attribute in a relation, for example:

```
MIN (Salary) FROM EMPLOYEES
```

This aggregate would return a single value containing the smallest of all the Salary values in the Employees relation. Aggregates can also be computed on portions of relations by simply adding a qualification clause to identify the portion of the relation desired. Queries containing aggregates are the only ones requiring processing by the controller, and may be viewed as occurring in two phases. In Phase I, which is accomplished by the RCs in parallel, partial aggregate results are computed on the portions of the database managed by each RC and these partial results are sent to the controller. Phase II begins when the controller receives all of the partial results from the RCs, and consists of a re-application of the aggregate function to the set of partial results. The aggregate functions are described in more detail in the following sections.

Min and Max. Calculating the minimum (Min) and maximum (Max) values of an attribute, in a set of records, requires a Project

operation to be performed on the desired set of records. Once the values have been identified by the projection each RC extracts only the minimum or maximum value, from the set, and transmits it to the controller. The controller, after receiving answers from all the RCs, performs the min/max calculation on the set of partial results and transmits the final answer to the host.

Count, Sum, and Average. The aggregate function Count returns the number of tuples in a designated set. The set can be a relation or a portion of a relation specified by a qualification clause. In order to isolate the records to be counted a Select operation is performed, on the relation, based upon the predicates in the qualification clause. Once qualifying records are identified and located they are counted and the total is transmitted to the controller. After receiving totals from all RCs the controller adds them yielding the total count desired.

Unlike the Count function, which tallies records, the Sum function is used to add the values of an attribute in a relation or a subset of a relation. First, a qualified Project operation, of the desired attribute on the relation, isolates the attribute values. Next, the values are added and the sum is sent, from each RC, to the controller. After receiving sums from each RC the controller simply adds them and returns the total to the host.

The last aggregate operation, Average, combines the actions of the Sum and Count aggregate functions to compute the average value of an attribute for a relation or a portion of a relation. A qualified Project operation isolates the desired attribute values in the secondary storage of the RCs. The number of records identified is counted (Count) and then all of the values are added (Sum). The resultant count and sum are transmitted to the controller. The controller, after receiving a sum and a count from each of the RCs, adds all of the counts to produce a final count and adds all of the sums to produce a final sum. In the final step the controller calculates the average by dividing the final sum by the final count and sends the result to the host.

Two-Relation Queries

Operations on two relations, supported by RRDS, include Union, Difference, Intersection, and Join. These queries are processed in the same manner as one-relation queries by the controller. More processing is required by the RCs, however, and is described in the following sections.

The Union Operation. In RRDS, two relations with the same schema can be combined through a Union operation. A union between relations R1 and R2 is specified as follows:

R1 UNION R2

When receiving a Union command, from the controller, each RC begins by consulting the directory to determine the records in

each relation and their addresses in secondary memory. Once the records are identified and located, each RC performs the union operation between its portion of R1 and R2 and sends the result to the controller. After receiving results from all RCs, the controller collects them into a temporary relation, eliminates duplicate records, and transmits the final result to the host.

The Difference and Intersection Operations. Difference operations are used to identify records of one relation which are not contained in another relation. For example, the command:

R1 DIFF R2

will return the records in R1 which do not exist in R2. As in the Union operation, this command may only be performed on relations with a common schema. When receiving a difference command, from the controller, each RC begins by consulting the directory to determine the records in R2 and their addresses in secondary memory. Each RC then broadcasts its portion of this relation to all the other RCs. Next, each RC consults the directory and locates the records of its portion of R1. Now, each RC has its portion of R1 and the entire R2. Each RC then performs the difference operation between its portion of R1 and the entire R2, and sends the result to the controller. Each RC then discards the portion of R2 which it did not originally manage. After receiving results from all RCs, the controller collects them into a temporary relation, and transmits the final result to the host.

Intersections are used to identify records which are common to two relations with identical schemas. In RRDS the command:

R1 INTSECT R2

returns only those records appearing in both R1 and R2. When receiving an intersect command, from the controller, each RC begins by consulting the directory to determine the records in each relation and their addresses in secondary memory. Once the records are identified and located the RCs must determine which relation is smaller, in terms of records. Each RC then broadcasts its portion of this relation to all the other RCs. After broadcast, each RC has its portion of the larger relation and the entire smaller relation. After receiving the smaller relation (say, R2) from the other RCs, each RC selects only those records contained in both its portion of R1 and the entire R2, and sends the result to the controller. As in the other two-relation operations, the controller collects the results into a temporary relation and sends the result to the host.

The Join Operation. The result of a Join operation, over some common attributes, is a new relation with all of the attributes of the original relations, and the information from those records which have the same value for the common attributes. Unlike the two-relation operations previously described, two relations to be joined need not have the same schema but only one, or more, attributes in common. An example join command in RRDS is shown below:

R1 JOIN R2

RC processing occurs exactly as in the intersect operation to the point where each RC has its portion of R1 and the entire R2 (the smaller relation in our example). At this point each RC forms a temporary relation containing the join of its portion of R1 and the entire R2. The join algorithm assumed for this phase of RRDS design is the straightforward serial join (nested loop) approach. After the join is accomplished at the RCs the temporary relations are sent to the controller which forwards the result to the host.

Performance Analysis

The query processing strategies described in the previous section have been incorporated into an RRDS predictive performance model. The model, which simulates the sequence of events from the time a query enters RRDS to the time the results are transmitted to the host, includes the data access, placement, and directory management strategies developed in the previous chapters. A workload model has been implemented to allow variance of database, user, and query characteristics in order to simulate different operating environments for the system. In the experimentation phase, workload conditions were altered and the effects on RRDS performance observed. This process, from model development through results collection and analysis, is described in this section.

Performance Analysis Goals, Methodology, and Approach

As stated in Chapter 3, the ultimate design goal for RRDS development is performance proportional to the number of

processing elements. Throughout each stage of the design process certain decisions have been made with this goal in mind, starting with development of the preliminary architecture. The first goal of the performance analysis is to predict if proportional performance can be realized by our proposed design. In addition, the performance evaluation should provide valuable information about the behavior of the system, under certain specific conditions, spawning ideas for future research. Strengths and weaknesses of the design will be revealed as well as favorable and adverse operating environments for RRDS. Valuable information such as which operating environments create bottlenecks in certain RRDS components will provide insight into the best applications for the system. Finally, the performance analysis will provide predicted performance parameters with which future RRDS prototypes may be compared. Conversely, the results of the performance analysis can be verified, and the simulation model validated, by the actual performance of implemented prototypes.

The methodology employed for conducting the performance analysis is to run a set of experiments on SLAM simulation models of RRDS. The simulation system combines a parametric workload model with open queuing network representations of various system configurations. Statistics are collected on various system performance parameters such as query response time, for each query type, and RRDS component utilization. In order to better present the results, and ascertain the extent to which the design

goals are satisfied, a performance metric called the percentage ideal goal (Hsiao 1981b) is utilized. The response time of a baseline 2-RC RRDS serves as the reference point. The percentage ideal goal of RRDS utilizing n RCs is then defined as the following ratio:

$$\frac{2(\text{Response Time of RRDS With 2 RCs}) (100)}{n(\text{Response Time of RRDS With } n \text{ RCs})}$$

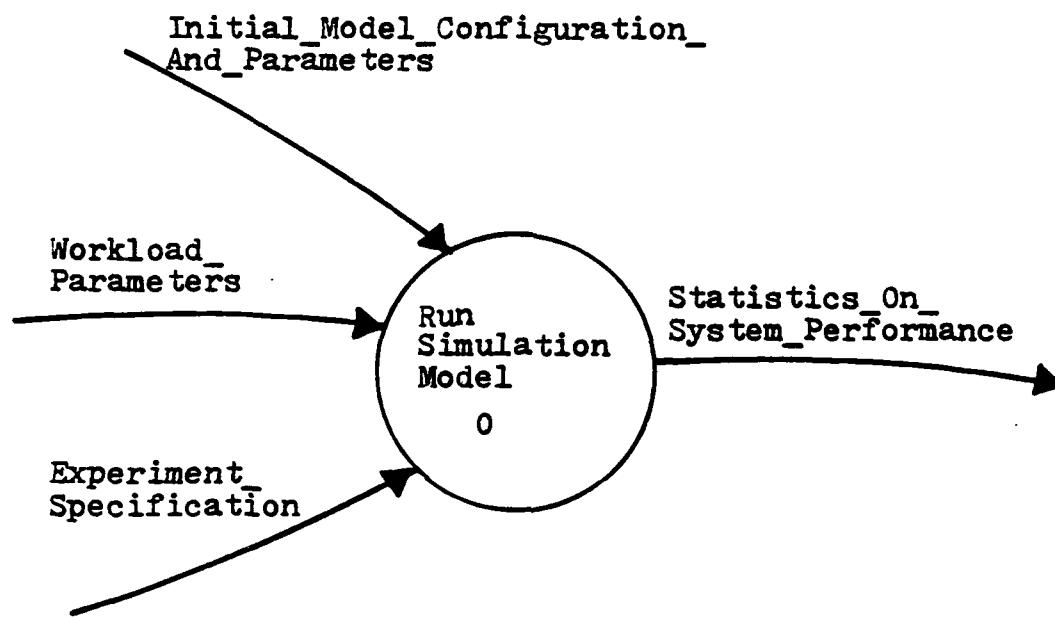
With the above definition the percentage ideal goal of RRDS with, say, four RCs ($n = 4$) will be 100 only if the response time of the system with four RCs is exactly half the response time of RRDS with two RCs. Similarly, if the number of RCs is increased to six, the percentage ideal goal will be 100 only if the response time is exactly one-third of that of the 2-RC configuration.

The approach for model development is similar to the one described in Chapter 4 and employed at each level of system design. In this phase the subsystem models previously developed are incorporated into the RRDS system model. Three different RRDS simulation models have been constructed: 2-RC, 4-RC, and 6-RC systems. The performance study requires execution of experiments on all three configurations, along with analysis of the percentage ideal goal results, for various workloads. Results of each experiment are collected, analyzed, and presented in tabular or graphic form, along with conclusions.

The RRDS Simulation Model

Based upon the process-oriented SLAM open queuing network simulation modeling paradigm, the RRDS simulation model consists of a set of SLAM networks, a hardware parametric model, and a set of workload parameters (the workload model) which can be varied to observe the effects on the system according to the experimentation plan. The experiments were performed on three versions of the RRDS model: 2-RC, 4-RC, and 6-RC configurations. The three versions were constructed by attaching RC-Ms components to the broadcast bus and redistributing the database. In the following sections a high-level DFD representation of the simulation model is provided along with explanations of the hardware parametric model, workload parameters, and the experimentation plan. The complete SLAM simulation model is provided in Appendix G of this dissertation.

The Queuing Network Model. A run of the RRDS simulation model is depicted in Figure 48, the DFD context diagram. The first input, describing the initial system configuration, consists of factors such as starting conditions, initialization of random number streams, and specifications of statistics to be collected. The workload parameters describe the database characteristics (number of records per relation, number of attributes per relation, etc.), query characteristics (query types, query composition, etc.), and user characteristics (query interarrival times, response set sizes, etc.). These parameters represent the system environment for the simulation. The experiment specification



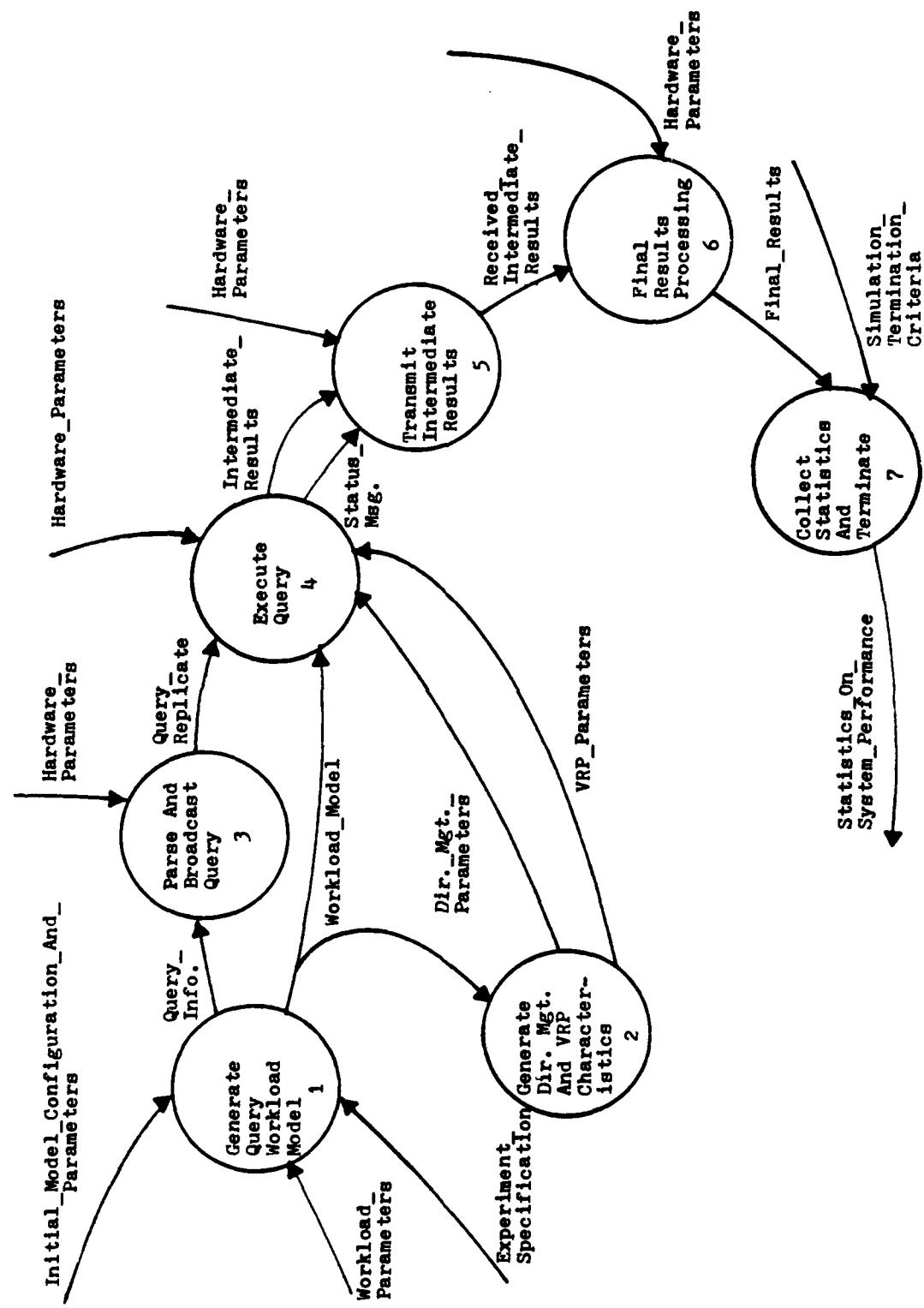
Context Diagram

Figure 48. RRDS Simulation Model Approach Context

provides the scenario for each simulation. Since each experiment is designed to observe the effect(s) of altering certain conditions, the experiment specification dictates which variables will be altered and which results will be observed. The workload parameter values are, therefore, a function of the experiment specification. Running the RRDS simulation model will provide statistics such as response time for each query type, average system response time, throughput, RRDS component utilization, queue wait times, queue lengths, and service utilization figures.

An RRDS simulation consists of the seven steps depicted in Figure 49. The front-end of the queuing network actually consists of a subnetwork of SLAM Generate, Assign, and Event nodes which, based upon the workload parameters and experiment specification, generate the query information, workload model, directory management parameters, and VRP parameters for the system. A variable number of any, or all, of the query types modeled can be generated at desired interarrival times, beginning at a specified initial creation time. When multiple queries of a certain type are generated, interarrival times are assumed to be Poisson with the mean specified as part of the workload model.

The first processing step after the generation phase is the parse and broadcast of queries to the RC network. All queries are parsed and checked for correctness by the RRDS controller. They are subsequently broadcast to the RCs, for processing, on the bus. The hardware parameters dictate the duration of activities such as parsing and query broadcast. These parameters



will affect all phases of processing until final results are produced. Query broadcast from the controller generates replicas of the query for delivery to all the RCs, and subsequent use in the Query Execution portion of the model. The bus, as well as all other system components, are assumed to be 100 percent reliable. We also assume that all messages are received intact and in the order sent. All system components are modeled as SLAM resources facilitating collection of statistics on component utilization.

The Query Execution portion of the simulation model, illustrated in Figure 50, is the heart of the system. This portion of the model represents the RC actions for each query type, according to the algorithms discussed in chapters 5 through 7 and Section 8.1 of this chapter. In the SLAM network each query type is modeled on a separate branch of queues and services. We chose to model the five query types shown in Figure 50, instead of all the relational operations available in RRDS, because the characteristics of all the operations are contained in these five query types. Hardware, directory management, VRP, and workload parameters all affect the query execution portion of the simulation model. During this portion of the simulation statistics are collected on bus, RC, and Ms utilization as queries enter the queues for inter-RC transmission, I/O, RC record processing, and transmission of results to the controller.

Figure 51 shows the first decomposition of query execution for Select queries. The first step is to access the B+_tree

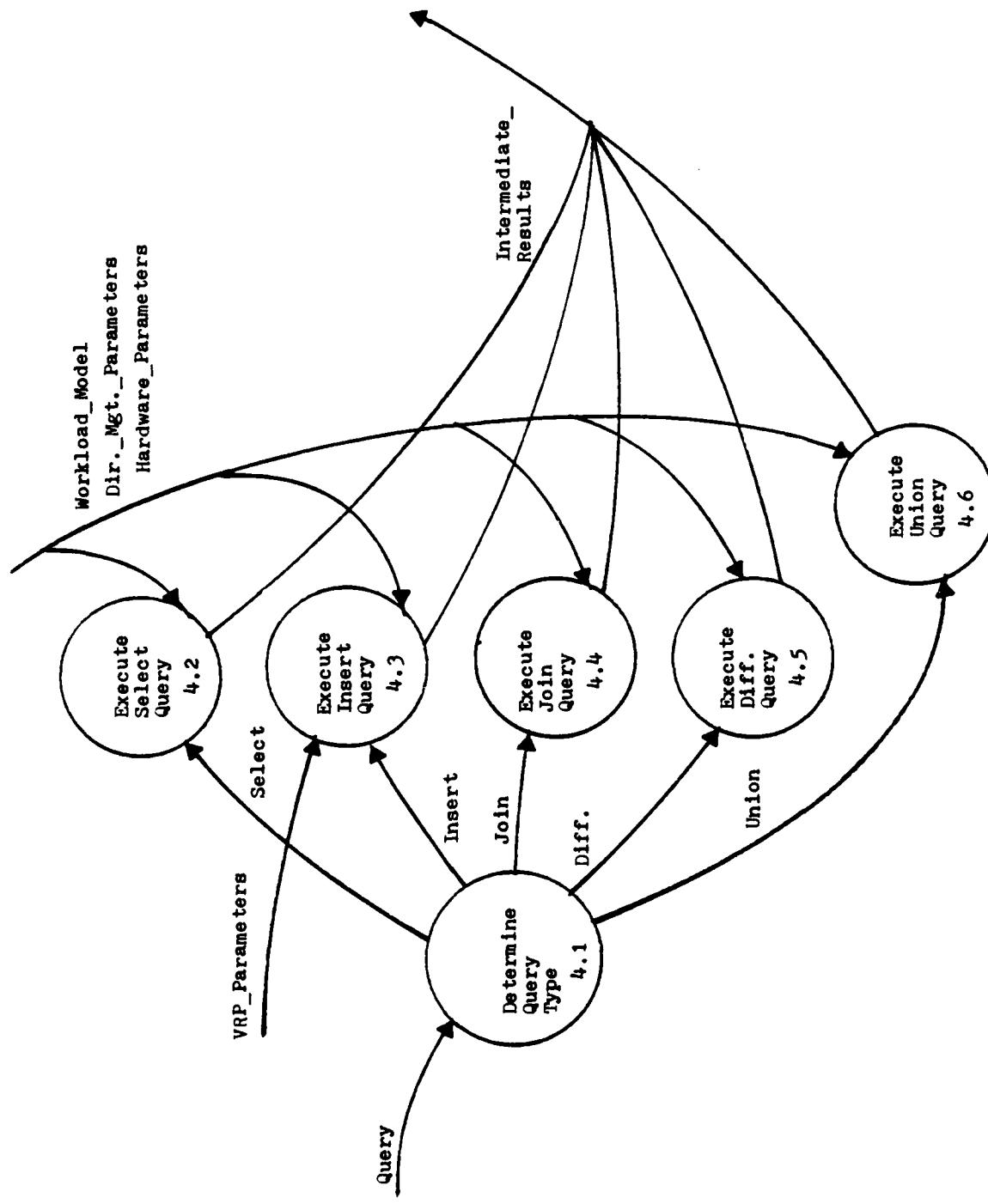
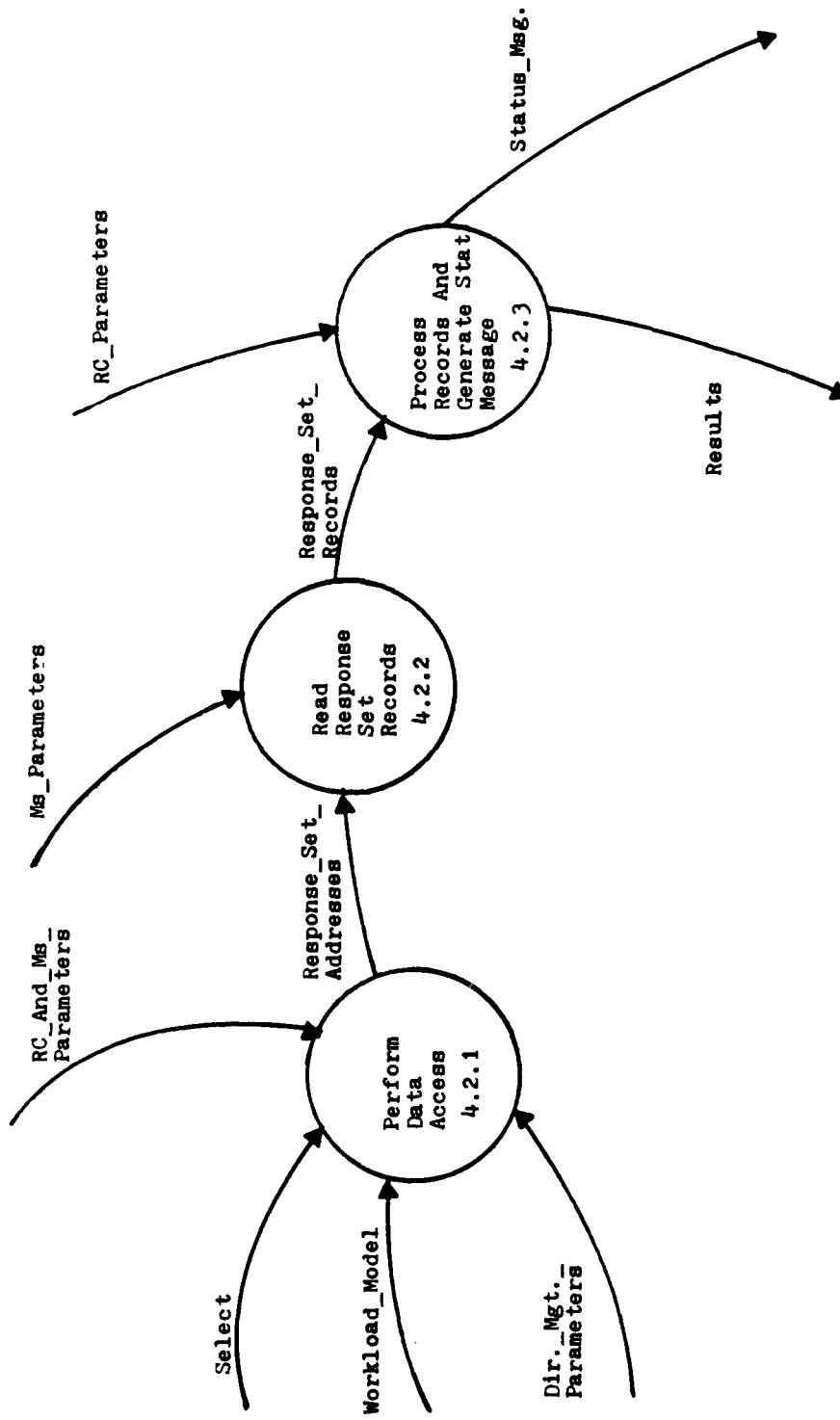


Figure 50. ^{4.1} Execute Query Execution Portion Of Simulation Model (2nd Decomposition)

directory and perform record address generation. Once the records comprising the query response are known they are read from the secondary memory in step two. Finally, record processing is accomplished in the RCs, identifying the records which satisfy all the query predicates (more specifically, predicates on non-directory attributes). The results of the Select query execution are a status message indicating that the response set has been computed and is ready for transmission to the controller, and the set of records satisfying the query. (The query execution diagrams for the other four types are similar, in concept, to the Select query execution and, therefore, will not be presented.)

For all query types the result of query execution is the intermediate response set records and/or a status message, to be transmitted to the controller for final processing and subsequent transmission to the host. As shown in Figure 49, the next step is Intermediate Results Transmission. As each RC finishes processing on its portion of the database for a certain query it transmits the intermediate result to the controller via the bus. Once the controller has received the intermediate results from all the RCs (modeled in SLAM with a Match node) it performs Final Results Processing, such as final duplicate elimination for a Union query. This is the final step for RRDS and, once the controller has finished, statistics on query time in system are collected and the query entity is destroyed.



4.2. Execute Select Query

Figure 51. DFD For Select Query Execution

The Hardware Parametric Model. RRDS consists of a controller attached to a number of RCs via a broadcast bus which facilitates communication between the controller and the RC network as well as inter-RC communication. In addition, each RC has a number of dedicated disk drives which we have called secondary memory (Ms). In the simulation models it is assumed that the controller and RCs are all 1 MIP processors. The disk drive parameters are those of the Fujitsu Eagle device and the broadcast bus is assumed to transmit at a rate of 2Mbytes/second. The hardware parametric model, based upon the operating characteristics of these components, is given in Table 23.

The Workload Model. The workload model facilitates simulation of RRDS for different operating environments based upon database, user, and query characteristics. Values specified in a workload parameter set are dictated by the goals and objectives of the experiment being performed. The workload model, constructed from these parameter values, allows variance of every important aspect of the operating environment, providing flexibility in experimentation options. The workload model, generated by the simulation model front-end, affects the directory management parameter set, the VRP parameter set for Insert queries, and all query execution activities. Table 24 lists the workload parameter set, from which the workload model is constructed for each simulation run.

Experimentation Plan. The workload model and experimentation plan were designed in concert to achieve the goals of the performance

TABLE 23. RRDS SIMULATION MODEL HARDWARE PARAMETERS

PARAMETER	VALUE	DESCRIPTION
Tb	18.000 ms	Time to read 512-byte block of Ms
Taddr	.280 ms	Time to read/write a record address from/to Ms
Tdid	.105 ms	Time to read/write a Descriptor identifier from/to Ms
Tdt	1.510 ms	Time to read a DT entry from Ms
Tread	3.510 ms	Time to read/write a record from/to Ms
Tparse	20.000 ms	Time to parse a query
Tgen	8.000 ms	Time to generate a status message
Tproc	.209 ms	Time to compare two records (as in Join or duplicate removal) or time to scan a record (as in Select with predicates on non-directory attributes)
Tbproc	.535 ms	Time to process a B+_tree node in the RC
Tdtproc	.090 ms	Time to process a DT entry in the RC
Tdidproc	.006 ms	Time to process a descriptor ID in the RC
Tadtrans	.004 ms	Time to transmit a record address on the bus
Trectrans	.050 ms	Time to transmit a record on the bus
Tmtrans	.025 ms	Time to transmit a status message on the bus

TABLE 24. THE WORKLOAD PARAMETER SET

PARAMETER	DESCRIPTION	ENVIRONMENTAL FACTOR
r	Average relation cardinality	Database
a	Number of attributes in a relation schema	Database
da	Number of directory attributes for a relation	Database
Pda	Probability for an attribute to be a directory attribute	Database
#RC	Number of RCs in RRDS	Database
Qtype	Query Type	User
Nsel	Number of Select queries	User
Nins	Number of Insert queries	User
Njoin	Number of Join queries	User
Ndif	Number of Difference queries	User
Nuni	Number of Union queries	User
IATsel	Interarrival time of Select queries	User
IATins	Interarrival time of Insert queries	User
IATjoin	Interarrival time of Join queries	User
IATdif	Interarrival time of Difference queries	User
IATuni	Interarrival time of Union queries	User
TFCsel	Time of creation of first Select query	User
TFCins	Time of creation of first Insert query	User
TFCjoin	Time of creation of first Join query	User

TFCdif	Time of creation of first Difference query	User
TFCuni	Time of creation of first Union query	User
Presp	Number of records in a predicate response set	User
%Result	Percentage of total possible response set comprising the actual query response set (i.e., a factor governing how much information the user actually wants)	User
Pa	Number of partitioning attributes for a relation	User
Ppa	Probability for an attribute to be a partitioning attribute	User
DT	Number of descriptors per partitioning attribute	User
c	Number of conjunctions in a query	Query
p	Number of predicates per conjunctions in a query	Query
Peq	Probability that a predicate is an equality predicate	Query

analysis. Experiments were developed to test for performance proportional to the number of RCs, identify bottlenecks, identify good and bad operating environments, and provide predictive performance parameters for comparison with system prototypes. An overview of the experimentation plan is provided in Table 25, describing each experiment in terms of the approach and overall goals. The workload parameter values, and the procedures for each experiment, are detailed in the next section.

TABLE 25. RRDS PERFORMANCE ANALYSIS EXPERIMENTATION PLAN

EXPERIMENT	GOALS
1. General Scenario Models: 2-RC/4-RC/6-RC Qtypes: Sel/Ins/Join/ Dif/Uni	<ul style="list-style-type: none"> - Predict average response time for each query type - Test for proportional performance using percentage ideal goal metric
2. Database Characteristics Models: 2-RC/4-RC/6-RC Qtypes: Sel/Ins/Join/ Dif/Uni	<ul style="list-style-type: none"> - Observe effect of relation cardinality on response time - Test for proportional performance using percentage ideal goal metric
3. User Characteristics Models: 2-RC/4-RC/6-RC Qtypes: Sel/Join Dif/Uni	<ul style="list-style-type: none"> - Observe impact of response set sizes on query response time - Determine impact on system performance of increased amounts of result data transmission - Test for proportional performance using percentage ideal goal metric
4. Query Characteristics Models: 2-RC/4-RC/6-RC Qtypes: Sel	<ul style="list-style-type: none"> - Observe effect on RRDS response time of varying number of predicates in Select queries - Test for proportional performance using percentage ideal goal metric
5. Query Characteristics Models: 2-RC/4-RC/6-RC	<ul style="list-style-type: none"> - Observe effect on RRDS response time of various predicate types in Select queries

Qtypes: Sel

- Test for proportional performance using percentage ideal goal metric

6. User Characteristics

Models: 2-RC/4-RC/6-RC

Qtypes: Sel/Ins/Join
Diff/Uni

- Observe impact of query interarrival time on response time and component utilization
- Identify system bottlenecks

7. User Characteristics

Models: 4-RC

Qtypes: Sel/Ins/Join/
Dif/Uni

- Determine best operating environment for RRDS in terms of query mix
- Observe effect on response time (system throughput) of various query mixes

8. Proportional Performance

Models: 2-RC/4-RC/6-RC

Qtypes: Sel/Ins/Join/
Dif/Uni

- Determine if proportional performance is achieved as relation cardinality and number of RCs are varied proportionally

9. Adding Ms Components

Models: 4-RC

Qtypes: Sel

- Observe effect of adding disk drives to RRDS running in an Ms-intensive mode

Experimentation And Results

In this section we present the results of executing the experimentation plan of Table 25 for various scenarios defined by the workload parameter set. A description of procedures and scenarios, along with results, is presented for each of the experiments in the experimentation plan. Because the RRDS model

is highly parameterized, the number of distinct scenarios is extremely large, making any exhaustive simulation infeasible. The experiments described herein represent an appropriately chosen subset of all possible cases, designed to achieve the goals of the performance analysis. In the first group of experiments (experiments 1 - 5) the goal is to observe the effects, on processing time, of varying the parameters which reflect database, user, and query characteristics. Next, the interarrival frequency and types of queries are varied to observe the effects of queuing on system performance and to obtain results on RRDS component utilization. The experimentation plan is completed by running simulations designed specifically to study the proportional performance criteria and observe the effects of increasing the number of disk drives per RC.

The results of running 100 queries of each type, for a general scenario, are shown in Table 26. These results represent an average response time for each of the five query types, on each of the RRDS configurations, over a range of workload parameter values. For this experiment query interarrival time was adjusted so that minimal queuing occurred, allowing observation of response time based on system processing overhead. Average relation cardinality ranged from 100 to 100,000 records for R1 and from 1 to 1,000 records for R2, with each record being fixed-length 100 bytes. Relation schemas consisted of between 2 and 10 attributes with the probability of an attribute being a directory, or partitioning, attribute ranging from 0 to 100.

TABLE 26. AVERAGE RESPONSE TIME FOR THE THREE RRDS CONFIGURATIONS

CONFIGURATION	QUERY TYPE				
	SELECT	INSERT	JOIN	DIFF	UNION
2-RC	99.53	.264	2855.4	2774.5	1671.3
4-RC	52.84	.186	1419.0	1377.1	619.0
6-RC	35.28	.180	949.7	910.2	415.6

percent. For directory management the B+_tree degree ranged from 10 to 100 keys per node (block of secondary storage). Select queries consisted of 1 to 3 conjunctions, each conjunction having up to three predicates. The probability that a predicate was an equality predicate (P_{eq}) ranged from 0 to 100 percent. For Insert queries, the number of descriptors per partitioning attribute, in VRP data placement, ranged from 0 to 4. Finally, the percentage of the total possible response set comprising the query response set (%Result) varied from .001 to .10. Each simulation run consisted of 100 queries, and a total of ten simulations were run for each query type, with the results averaged. Values for the workload parameters were selected from each range based upon a uniform distribution, indicating an equal probability for each value within the range.

Results of Experiment 1 indicate that RRDS will perform best for the one-relation operations Select and Insert, with average response time substantially higher for the two-relation operations Join, Difference, and Union. The higher response time

values for the two-relation queries are due to the increased volume of records involved, the fact that query processing time is a function of the product of the relations (Order $r_1 * r_2$), and duplicate removal (needed for Union queries) in the controller is required. The fact that the average response time for Difference operations is actually less than that for Joins is encouraging since, for Difference queries, the smallest relation is not necessarily broadcast between the RCs as in Joins.

The Percentage Ideal Goal figures from Experiment 1, presented in Table 27, are also encouraging, indicating that the design goal of performance proportional to the number of RCs is met or exceeded for Join, Difference, and Union query types. This is due to the fact that as more processors are introduced the size of relation fragments being processed is reduced proportionally. The 94 percent figure for Select queries is also satisfactory considering the fact that some serial processing is

TABLE 27. PERCENTAGE IDEAL GOAL FOR EXPERIMENT 1

CONFIGURATION

QUERY TYPE	4-RC RRDS	6-RC RRDS
Select	94.18	94.03
Insert	57.52	39.62
Join	100.73	100.22
Diff	100.73	98.36
Union	135.00	134.04

accomplished by the controller and bus. Finally, the non-proportional performance of Insert queries is expected since the actual insertion of a record is accomplished at only one RC. This result also indicates that VRP management overhead does not adversely affect performance. This is promising, especially since, for the performance analysis, all relations are assumed to be under VRP data placement regardless of their cardinality.

In the second experiment we were interested in observing the effects of relation cardinality (a database characteristic) on query response time, for each query type and for all three RRDS configurations. Simulations of 100 of each query type were run on relations with cardinalities chosen from three ranges: 100 - 500, 500 - 5000, and 5000 - 10000 records. Schemas for relations, in each cardinality range, consisted of ten attributes with three of them being directory attributes. VRP data placement was also accomplished on three partitioning attributes with four descriptors defined for each. Select queries consisted of one conjunction of three predicates, each predicate having an equal probability of being an equality or non-equality predicate. The %Result factor was set at .05 for all three cardinality ranges.

The results of Table 28 show that, as relation cardinality increased, so did response times for all query types. Each cardinality range represented an order of magnitude increase in the number of records comprising target relations. Response times for Select, Join, Difference, and Union queries increased

TABLE 28. EFFECT OF RELATION CARDINALITY ON RESPONSE TIME

RELATION CARDINALITY

CONFIGURATION	100 - 500	500 - 5000	5000 - 10000
2-RC	.627	4.97	12.49
4-RC	.330	2.41	6.34
6-RC	.242	1.65	4.28

Select Queries

RELATION CARDINALITY

CONFIGURATION	100 - 500	500 - 5000	5000 - 10000
2-RC	.105	.118	.124
4-RC	.072	.076	.078
6-RC	.061	.062	.063

Insert Queries

RELATION CARDINALITY

CONFIGURATION	100 - 500	500 - 5000	5000 - 10000
2-RC	10.28	777.4	5934
4-RC	5.22	397.2	3029
6-RC	3.53	270.3	2058

Join Queries

RELATION CARDINALITY

CONFIGURATION	100 - 500	500 - 5000	5000 - 10000
2-RC	10.08	759.7	5799
4-RC	5.04	379.7	2897
6-RC	3.36	253.0	1930

Difference Queries

RELATION CARDINALITY

CONFIGURATION	100 - 500	500 - 5000	5000 - 10000
2-RC	6.46	459.5	3488
4-RC	2.55	173.3	1308
6-RC	1.75	119.6	902

Union Queries

by a proportional amount, a good sign, showing the minimal effects of the serial portions of query processing. Performance for Insert queries was less affected by relation size because most processing overhead, for this query type, is dedicated to directory management activity instead of reading records from disk and performing operations on the records.

The percentage ideal goal figures for this experiment, given in Table 29, indicate performance gains similar to those observed in the previous experiment. The percentage ideal goal decreases slightly as more RCs are added to the system due to extra message traffic overhead and greater bus utilization, however this degradation does not appear to be significant.

An important parameter (a user characteristic), which could have a profound impact on response time, is the amount of information requested (%Result) by the user issuing the queries. This parameter impacts some serial processing actions of RRDS, particularly the transmission of results on the bus, to the controller, and the elimination of duplicate records, by the controller, for Union response sets. In Experiment 3 the %Result parameter was varied from 0 to 100%, across the following four ranges: 0% - 25%, 26% - 50%, 51% - 75%, and 76% - 100%. The database consisted of relations with 100 to 10000 records. The relation schemas, directory attributes, and VRP parameters were the same as the previous scenario. Simulations of three RRDS configurations were run for 100 of each of the four query types: Select, Join, Difference, and Union. Insert queries were not

TABLE 29. PERCENTAGE IDEAL GOAL FOR EXPERIMENT 2

CONFIGURATION

CARDINALITY RANGE	4-RC RRDS	6-RC RRDS
100 - 500	95.0	86.3
500 - 5000	103.2	100.6
5000 - 10000	98.5	97.2

Select Queries

CONFIGURATION

CARDINALITY RANGE	4-RC RRDS	6-RC RRDS
100 - 500	72.9	59.5
500 - 5000	77.6	63.4
5000 - 10000	79.4	65.6

Insert Queries

CONFIGURATION

CARDINALITY RANGE	4-RC RRDS	6-RC RRDS
100 - 500	98.5	97.1
500 - 5000	97.9	95.9
5000 - 10000	97.9	95.9

Join Queries

CONFIGURATION

CARDINALITY RANGE	4-RC RRDS	6-RC RRDS
100 - 500	100.0	100.0
500 - 5000	100.0	100.0
5000 - 10000	100.0	100.1

Difference Queries

CONFIGURATION

CARDINALITY RANGE	4-RC RRDS	6-RC RRDS
100 - 500	126.6	123.0
500 - 5000	132.5	128.1
5000 - 10000	133.3	128.9

Union Queries

considered in this experiment since they generate no response sets.

TABLE 30. EFFECT OF RESPONSE SET SIZE ON RESPONSE TIME

CONFIGURATION	%RESULT			
	0% - 25%	26% - 50%	51% - 75%	76% - 100%
2-RC	7.598	7.829	8.055	8.281
4-RC	3.905	4.135	4.362	4.588
6-RC	2.670	2.902	3.128	3.354

Select Queries

CONFIGURATION	%RESULT			
	0% - 25%	26% - 50%	51% - 75%	76% - 100%
2-RC	2793	3114	3429	3744
4-RC	1466	1787	2101	2417
6-RC	1021	1343	1658	1973

Join Queries

CONFIGURATION	%RESULT			
	0% - 25%	26% - 50%	51% - 75%	76% - 100%
2-RC	2643	2643	2643	2643
4-RC	1319	1319	1319	1319
6-RC	879	879	879	879

Difference Queries

CONFIGURATION	%RESULT			
	0% - 25%	26% - 50%	51% - 75%	76% - 100%
2-RC	1968	3311	4626	5941
4-RC	973	2315	3631	4946
6-RC	787	2130	3445	4760

Union Queries

The results illustrated in Table 30 indicate that the effects of having to transmit larger results, for Select queries, are minimal, e.g., a 300 percent increase in the size of the results set causes only a 10 percent increase in average response time on the 2-RC RRDS. Even when the number of RCs comprising the system is increased to six, resulting in the transmission of three times more messages, the average response time degradation (for the same increase in results set size) rises to only about 25 percent. This is an encouraging result indicating the bus will not create a bottleneck and, therefore, the absence of the communication network limitation problem. Similar results were observed for Join and Difference queries. The response time degradation for Join queries increased from approximately 34 percent to approximately 50 percent, as the number of RCs increased from two to six and for a 300 percent increase in results set size. As shown in the table, there was no degradation in performance of Difference queries. Union queries exhibited the most degradation in response time due to the removal of duplicate records (needed for Union) by the controller. As the response set, returned to the controller, grows so does the processing required to remove duplicates. Another factor contributing to the poor performance of Union queries, in this experiment, is the size of the results messages transmitted via the bus. For example, when %Result is 100 percent, it means each RC must transmit all of its records of the target relations. This also indicates that no duplicate removal is accomplished by the RCs, in parallel, on their portions of the

relations, causing the controller to perform all duplicate removal on the final results.

The percentage ideal goal figures, given in Table 31 for this experiment, even more graphically portray the impact of serial processing in RRDS. For Select, Join, and Union queries the percentage ideal goal drops steadily as %Result increases. Select performance still remains approximately proportional to the number of RCs. More of the advantage of adding processors is lost for Join queries as the size of messages, transmitted on the bus, increases. Finally, the detrimental effect of controller limitation is illustrated most profoundly for Union queries as the controller is forced to perform all duplicate record elimination when %Result is 100 percent.

Query characteristics, affecting RRDS processing, include the number and types of predicates in Select queries. The importance of these parameters lies in their direct impact on the amount of data access processing. Each predicate containing a directory attribute requires B+_tree access. Also, non-equality type predicates (e.g., SALARY < 10000) require more secondary storage access as multiple B+_tree leaf nodes are read and searched.

The first query characteristic parameter varied was the number of predicates, containing directory attributes, in Select queries. For 100 Selects the number of predicates was chosen from each of the four ranges: 1 - 3, 4 - 6, 7 - 9, and 10 - 12.

TABLE 31. PERCENTAGE IDEAL GOAL FOR EXPERIMENT 3

CONFIGURATION

%RESULT RANGE	4-RC RRDS	6-RC RRDS
0% - 25%	97.2	94.8
26% - 50%	94.7	89.9
51% - 75%	92.3	85.8
76% - 100%	90.2	82.3

Select Queries

CONFIGURATION

%RESULT RANGE	4-RC RRDS	6-RC RRDS
0% - 25%	95.2	91.2
26% - 50%	87.1	77.3
51% - 75%	81.6	68.9
76% - 100%	77.5	63.3

Join Queries

CONFIGURATION

%RESULT RANGE	4-RC RRDS	6-RC RRDS
0% - 25%	100.1	100.3
26% - 50%	100.1	100.3
51% - 75%	100.1	100.3
76% - 100%	100.1	100.3

Difference Queries

CONFIGURATION

%RESULT RANGE	4-RC RRDS	6-RC RRDS
0% - 25%	101.1	83.3
26% - 50%	71.5	51.8
51% - 75%	63.7	44.7
76% - 100%	60.0	41.6

Union Queries

Relation cardinalities and schemas were the same as the previous

experiment. The %Result factor was set at .05. Results, provided in Table 32, were not surprising, revealing a roughly proportional increase in response time to the increase in the number of predicates, for the 2-RC configuration. As the number of RCs comprising the system increases the same relative performance is observed, indicating that the bus will not be a system bottleneck.

Percentage ideal goal figures for this experiment, provided in Table 33, indicate proportional performance gains as the number of RCs increases, further illustrating the benefits of parallelism. The results of Experiment 4 indicate that the complexity of the queries issued can be successfully offset by expanding the system, an indication of RRDS extensibility.

In Experiment 5 the second query characteristic parameter, predicate type, was varied to observe its impact on Select response time. For Select queries, consisting of three predicates, the probability that a predicate is an equality

TABLE 32. EFFECT OF NUMBER OF PREDICATES ON SELECT RESPONSE TIME
NUMBER OF PREDICATES IN SELECT QUERY

CONFIGURATION	1 - 3	4 - 6	7 - 9	10 - 12
2-RC	5.118	12.900	20.500	28.190
4-RC	2.614	6.526	10.440	14.350
6-RC	1.778	4.426	7.075	9.724

TABLE 33. PERCENTAGE IDEAL GOAL FOR EXPERIMENT 4

CONFIGURATION

NUMBER OF PREDICATES	4-RC RRDS	6-RC RRDS
1 - 3	97.9	95.6
4 - 6	98.8	97.2
7 - 9	98.2	96.6
10 - 12	98.2	96.6

predicate (the least expensive type in terms of processing) was varied across the following ranges: 0 - .25, .26 - .50, .51 - .75, and .76 - 1.0. For each probability range, 100 queries were run on the same database as for the previous experiment. The effect of predicate type on response time is illustrated in Table 34. The predicate type has a major effect on Select response time due to the high degree of data access processing, on the B+_trees, required for non-equality predicates. As Peq increased by a factor of 300 percent the average response time for Select

TABLE 34. EFFECT OF PREDICATE TYPE ON SELECT RESPONSE TIME

CONFIGURATION	Peq			
	0 - .25	.26 - .50	.51 - .75	.76 - 1.0
2-RC	13.27	9.57	5.77	1.98
4-RC	8.81	4.86	2.96	1.05
6-RC	4.60	3.29	2.02	.74

queries dropped by a factor of 600 percent to 700 percent depending upon the RRDS configuration. The percentage ideal goal figures for this experiment, given in Table 35, also indicate that the processing overhead (for this case, the overhead associated with non-equality predicates) can be offset by adding more RCs to RRDS.

In the last three experiments we explored the effects of certain parameters on query response time. figures 52 through 56, a set of composite results graphs, depict the relative impact of these parameters by query type. In each figure the percentage increase in query response time is plotted against the percentage increase in values for each applicable parameter. Different parameters apply to different query types (e.g., the response set and predicate type parameters apply to Select but not Insert queries). These results illustrate which parameters have the most impact on each query type. Select queries are more affected by the query characteristics defining the number and type of

TABLE 35. PERCENTAGE IDEAL GOAL FOR EXPERIMENT 5

CONFIGURATION

Peq	4-RC RRDS	6-RC RRDS
0 - .25	97.5	96.3
.25 - .50	98.5	96.9
.51 - .75	97.6	95.4
.76 - 1.0	94.1	89.2

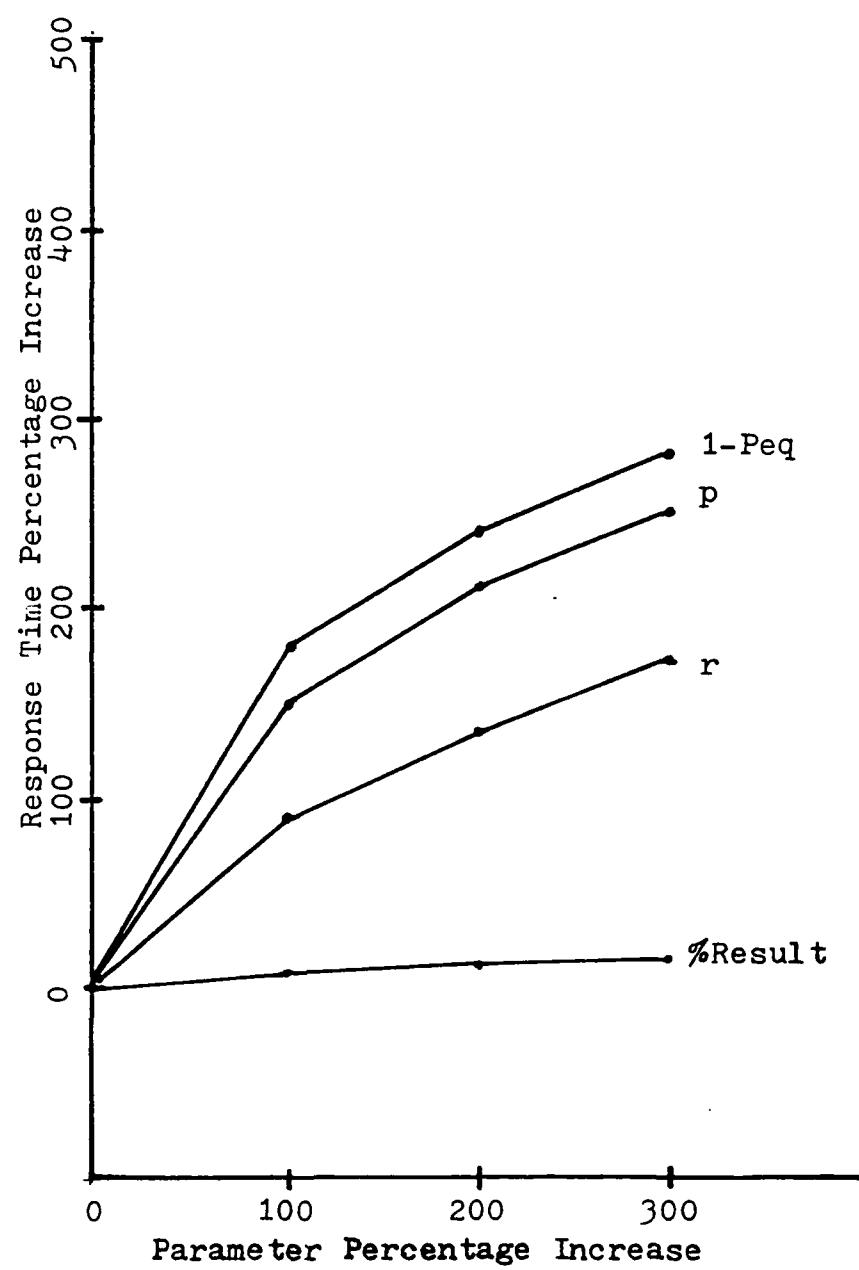


Figure 52. Impact Of Different Parameters On Select Response Time

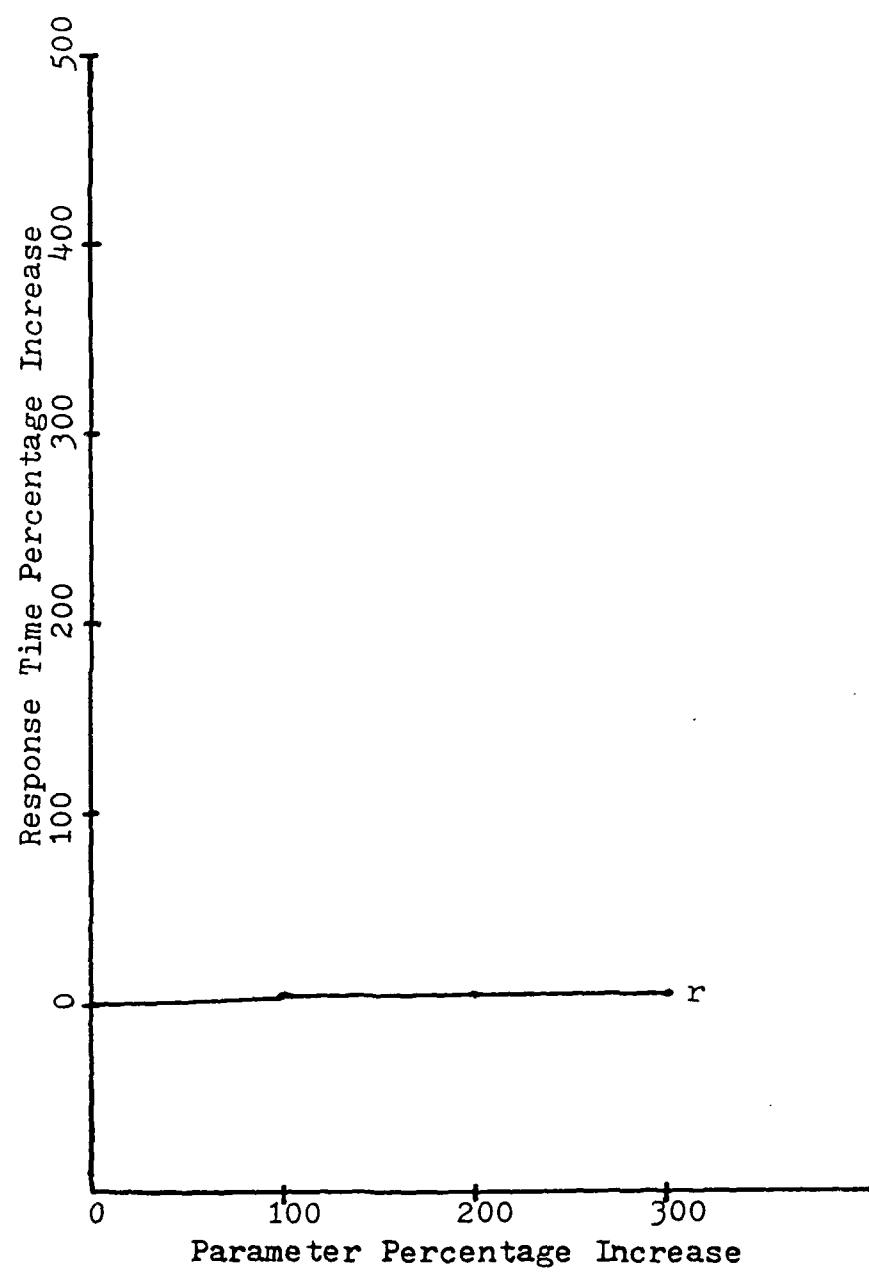


Figure 53. Impact Of Different Parameters On Insert Response Time

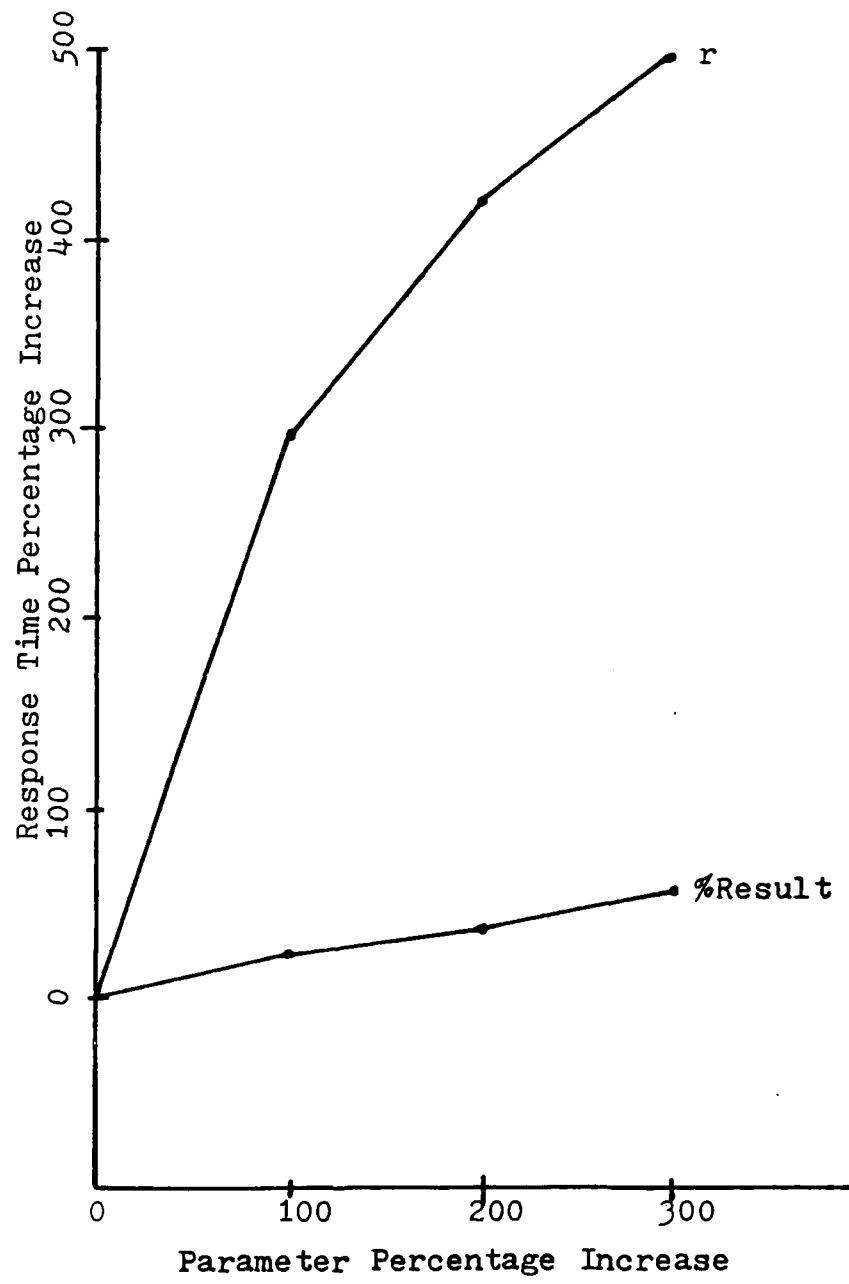


Figure 54. Impact Of Different Parameters On Join Response Time

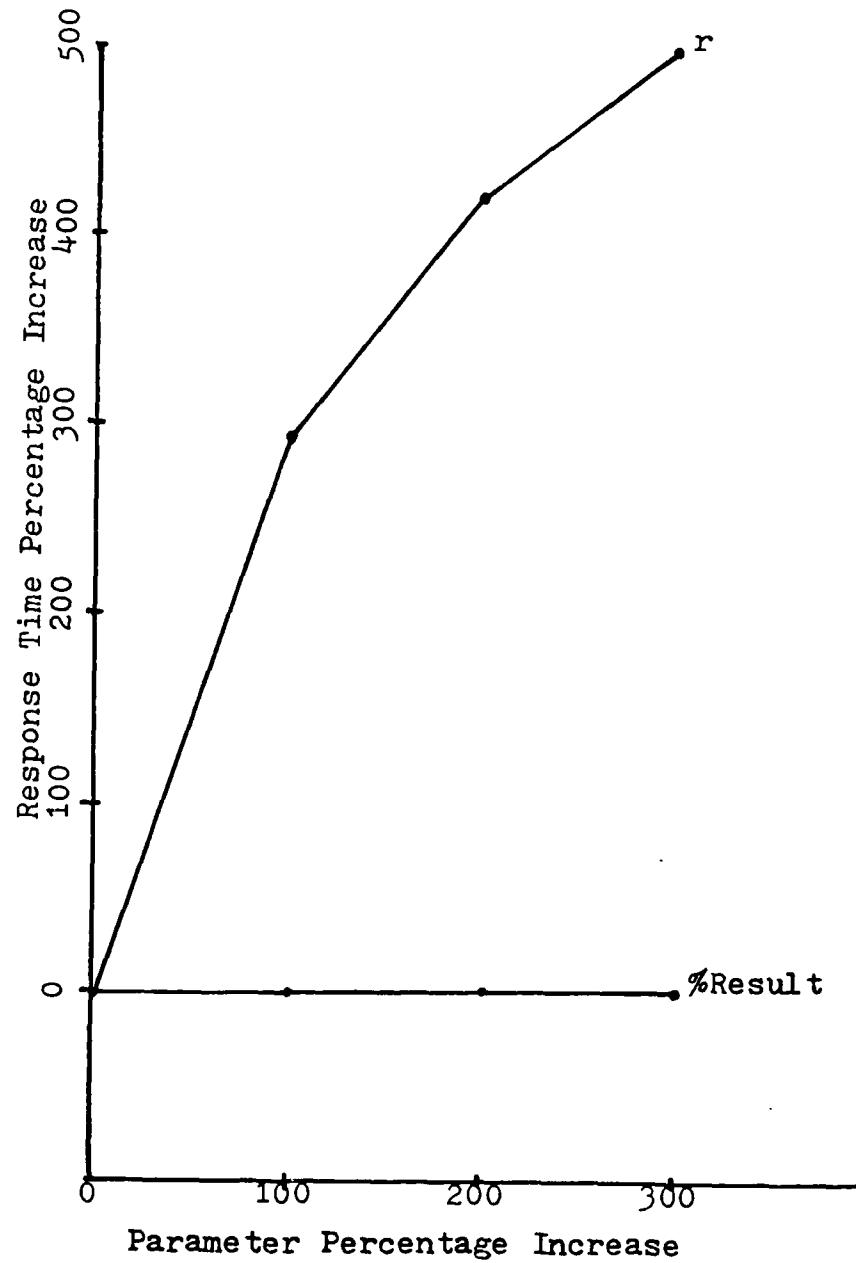


Figure 55. Impact Of Different Parameters On Difference Response Time

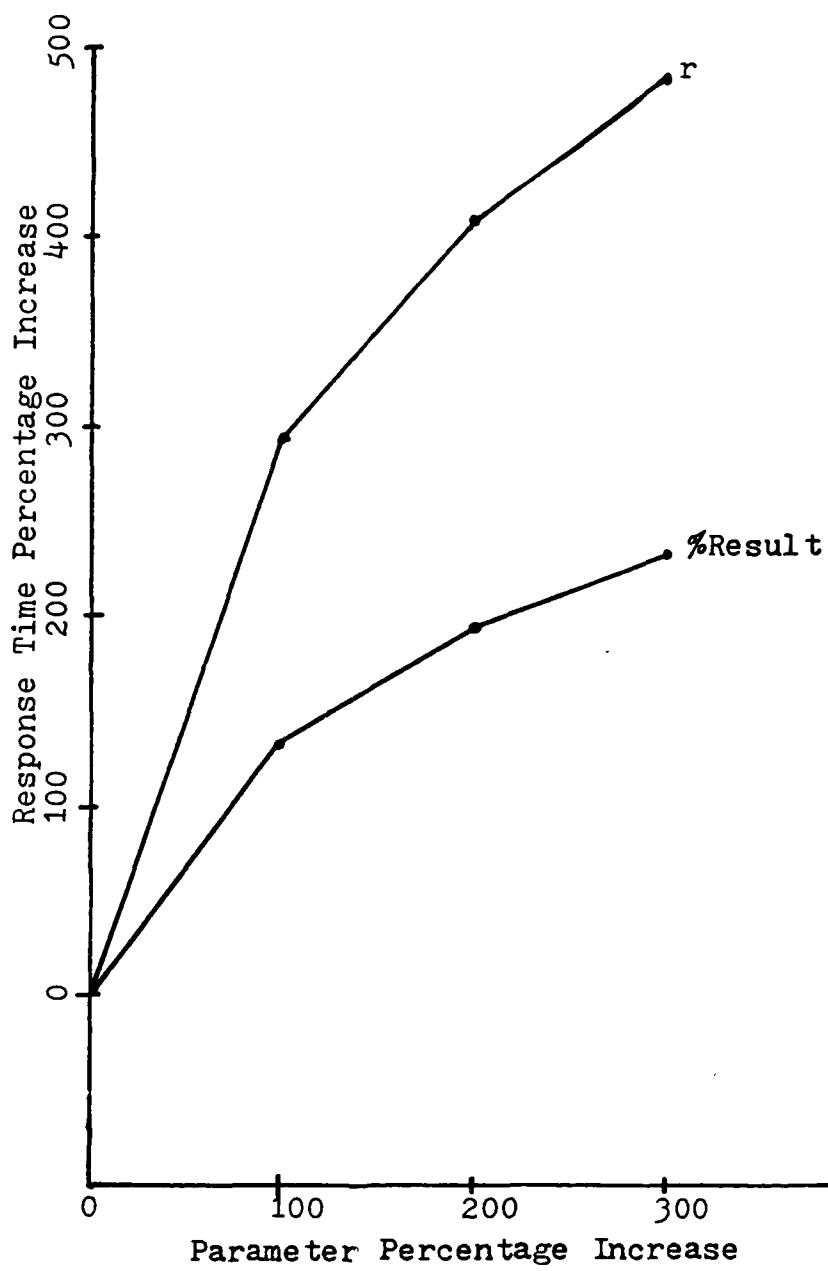


Figure 56. Impact Of Different Parameters On Union Response Time

predicates, than by relation cardinality or the response set size. On the other hand, Insert queries are virtually unaffected by changes in relation cardinality and the two-relation queries (Join, Difference, and Union) are most heavily impacted by the cardinality of the target relations. The size of the response set becomes more important for Union queries due to duplicate record removal. The information presented in these graphs is valuable in determining the best applications and operating environments for RRDS.

Up to this point, our results have been for scenarios where the query interarrival times were adjusted such that queuing in the system had a minimal effect on the average response time. The objective was to observe the effects of varying different parameters on the system. Therefore, response time figures reflected RRDS processing time based on ranges of values for certain variable(s). We must, however, investigate the effects of queuing in the system. One of the important parameters (user characteristic) of the RRDS workload model is the frequency at which queries, of the various types, enter the system. In Experiment 6 the impact of query interarrival rate, and the resultant queuing effects on response time, for each query type were investigated. Determining potential system bottlenecks, through examination of component utilization figures, was also a primary objective of this phase of the performance analysis.

For a database consisting of relations with 100 to 10000 records each, and schemas consisting of ten attributes with three

directory and partitioning attributes, 100 queries of each type were run for various interarrival frequencies. For each query type the interarrival time (IAT) was varied such that system performance could be observed, from a scenario with no queuing to a scenario where RRDS approaches saturation. The value ranges for interarrival times were derived using the 2-RC RRDS configuration. Then, the same scenarios were run on 4-RC and 6-RC systems to observe how well adding RCs compensates for throughput degradation due to queuing. The first results, presented in the graphs of figures 57 through 61, are the average response times for each query type as a function of interarrival times. Two conclusions can be drawn from these results: 1) query interarrival time has more impact on average response time than any other parameter observed so far, and 2) the adverse effects of shorter interarrival times can be successfully overcome by adding more RCs to the system - another indication of RRDS extensibility. On each graph there is a point where the 2-RC version of RRDS becomes saturated and throughput ceases to improve. At this point average query response time degrades quickly. For example, in Figure 57, average response time begins to increase quickly at the point where Select queries are arriving at a rate between one every five and one every ten seconds. At this point some component(s), in the system, become saturated causing excessive queuing delays. We also note that as RCs are added to the system the saturation point shifts left and better-than-proportional performance is realized (proportional performance would be indicated by curves with the same slope).

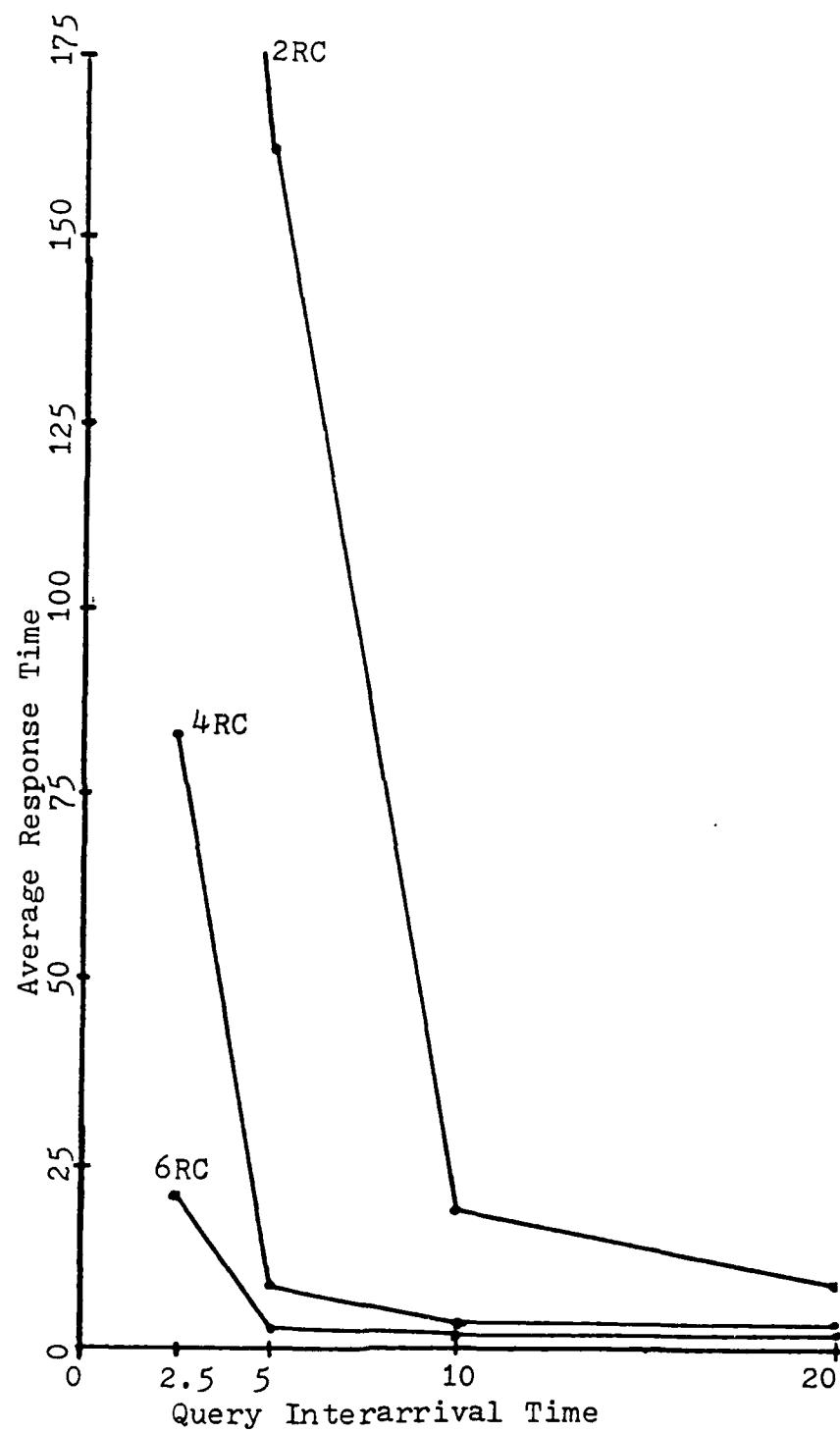


Figure 57. Effect Of IAT On Select Response Time

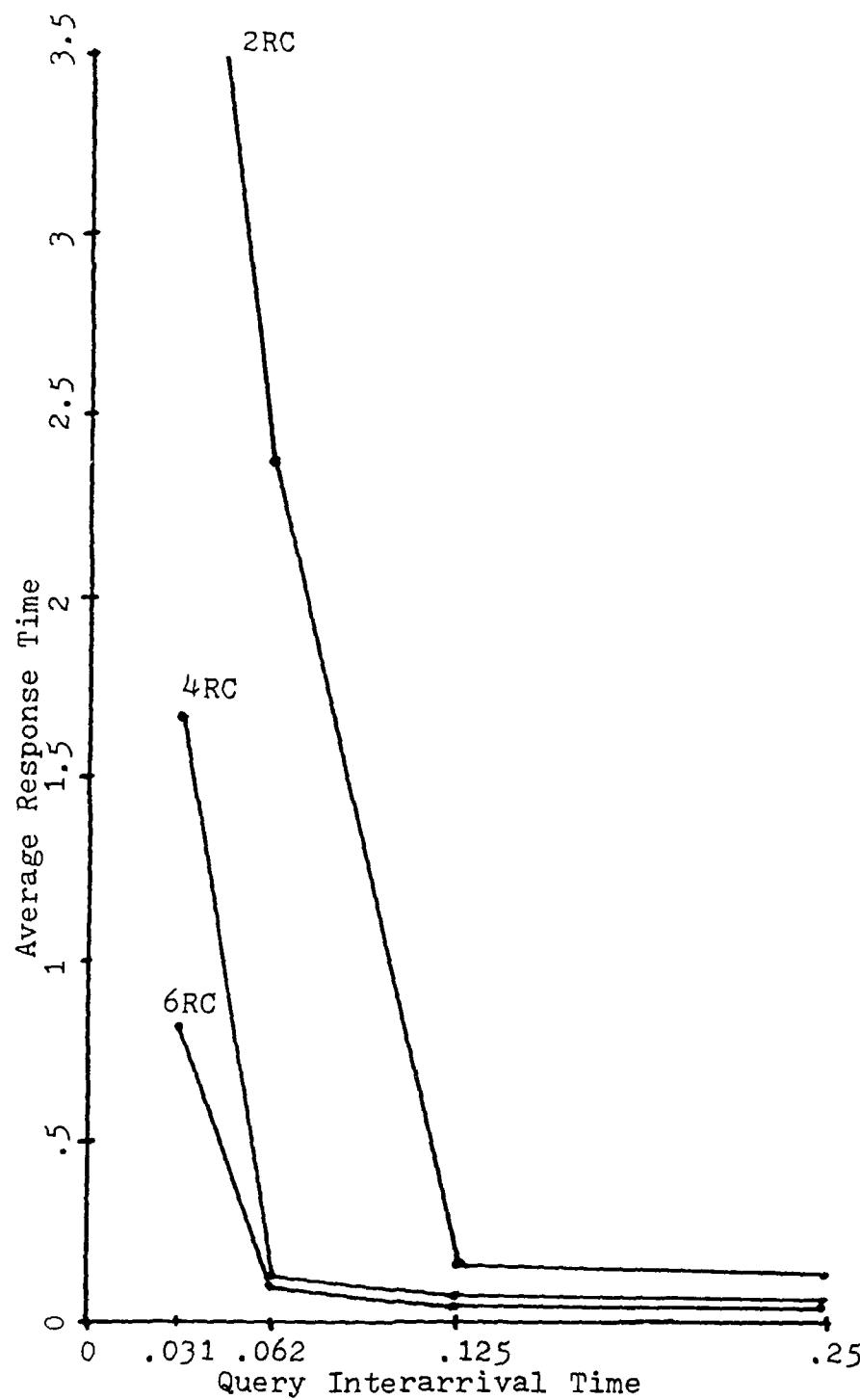


Figure 58. Effect Of IAT On Insert Response Time

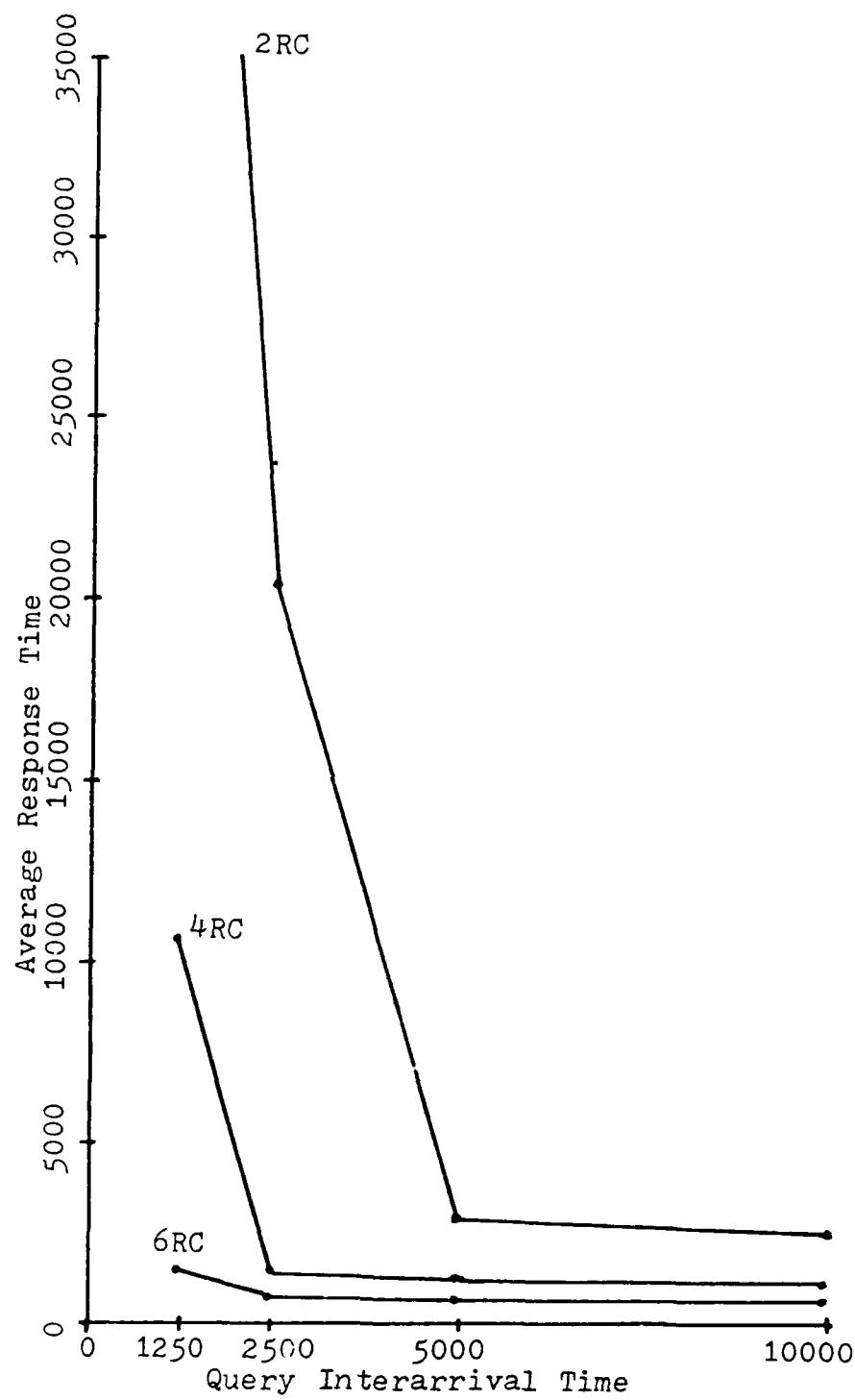


Figure 59. Effect Of IAT On Join Response Time

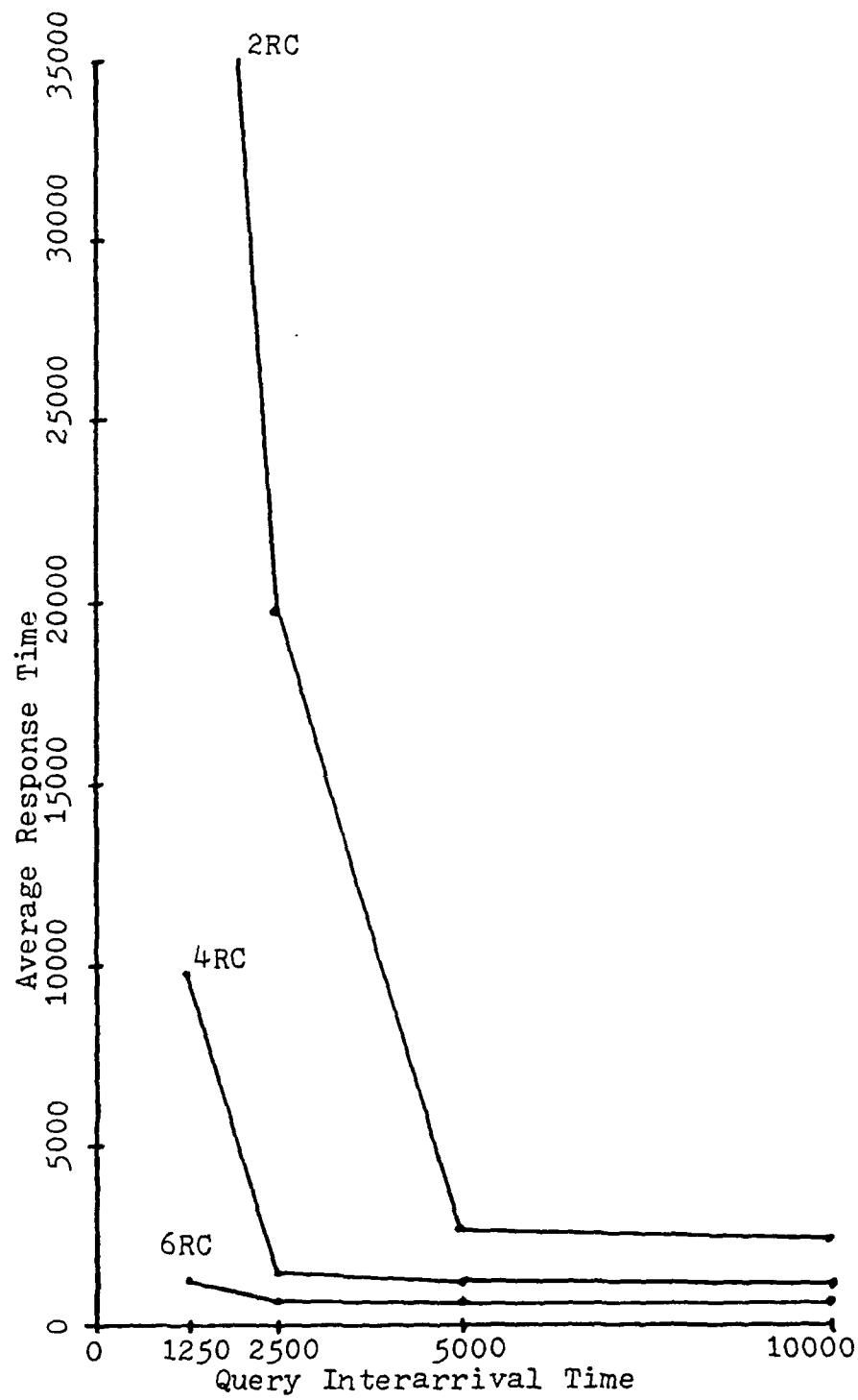


Figure 60. Effect Of IAT On Difference Response Time

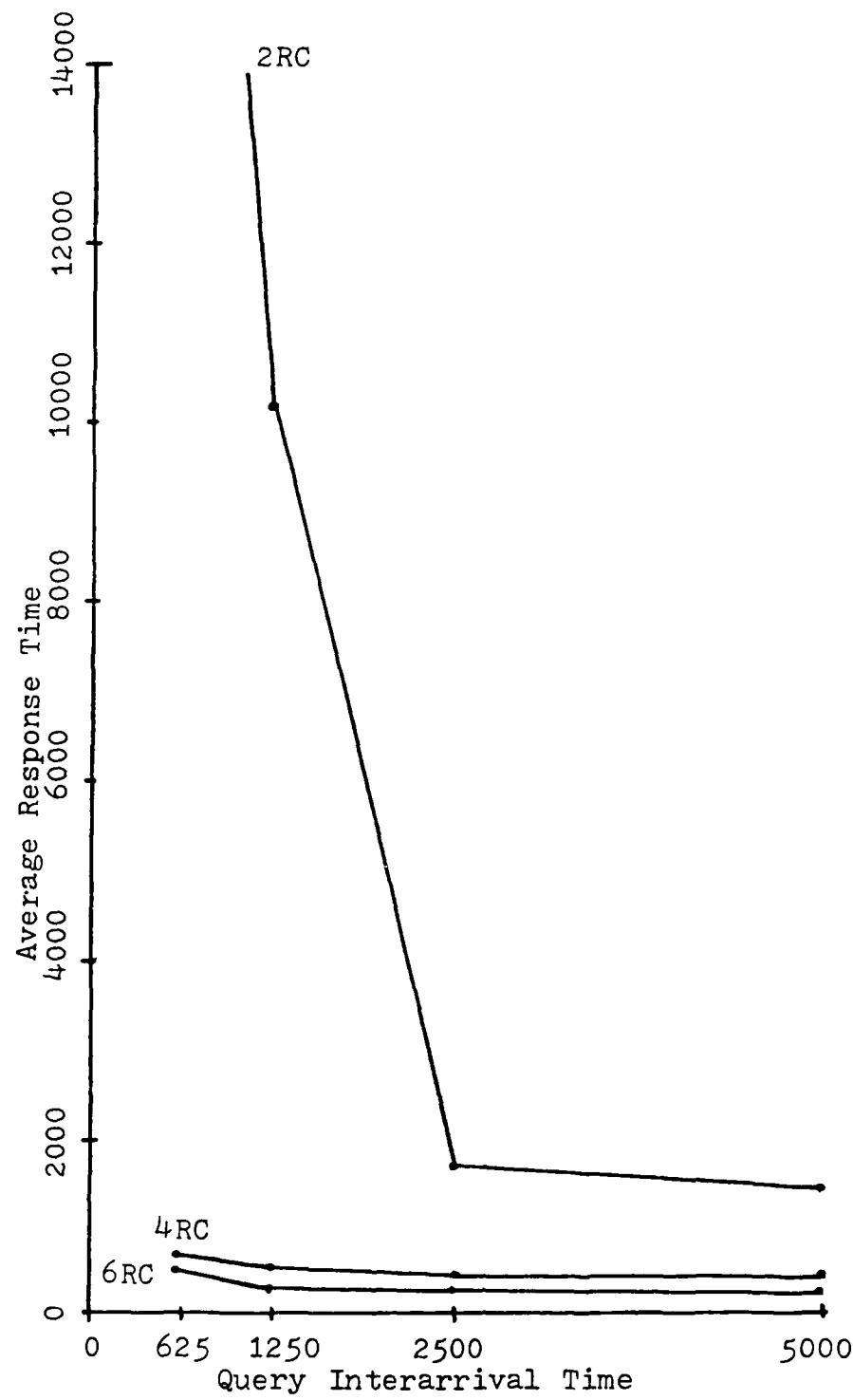


Figure 61. Effect Of IAT On Union Response Time

Additionally, not only does the saturation point change with the query type, the degree of performance improvement with additional RCs also changes. The extent to which performance gains are achieved, in a system approaching saturation, is seen in the percentage ideal goal figures of Table 36. For Select and Insert queries, as the 2-RC system saturates, the benefits of the 4-RC and 6-RC configurations peak. The results for Join, Difference, and Union queries are even better, with the performance of the 6-RC RRDS continuing to improve even after the 4-RC configuration begins to degrade. In all cases the results indicate that the throughput degradation, due to decreased query interarrival time, can be alleviated by adding more RCs to the system. Finally, from the interarrival time vs average response time graphs, it can be seen that Insert queries are most sensitive to IAT changes. This is also useful information when considering possible applications for RRDS.

Other useful information, revealed during this experiment, concerns isolation of system bottlenecks. In the graphs presented earlier it was observed that at some point each system reached saturation. We are now interested in determining why this occurred. By collecting statistics on RRDS component utilization, for each interarrival time scenario, components creating bottlenecks can be identified for each query type. This valuable information will provide insight into the best operating environments and applications for the system as well as indicating areas where performance can be improved by adding more

TABLE 36. PERCENTAGE IDEAL GOAL FOR EXPERIMENT 6

CONFIGURATION

SELECT IAT	4-RC RRDS	6-RC RRDS
40.0	98.2	96.6
20.0	104.1	102.4
10.0	200.1	208.0
5.0	863.8	1467.0
2.5	172.3	458.9

Select Queries

CONFIGURATION

INSERT IAT	4-RC RRDS	6-RC RRDS
.50	77.6	63.0
.25	78.2	63.5
.125	83.9	69.9
.063	857.9	771.2
.031	139.9	187.8

Insert Queries

CONFIGURATION

JOIN IAT	4-RC RRDS	6-RC RRDS
20000	97.8	95.7
10000	97.8	98.9
5000	112.6	110.2
2500	654.6	738.9
1250	358.9	1586.2

Join Queries

CONFIGURATION

DIFFERENCE IAT	4-RC RRDS	6-RC RRDS
20000	100.0	100.0
10000	100.0	100.0
5000	113.3	113.3
2500	697.5	781.2
1250	380.0	1846.0

Difference Queries

CONFIGURATION

UNION IAT	4-RC RRDS	6-RC RRDS
10000	133.0	128.2
5000	133.4	128.7
2500	149.0	143.8
1250	831.1	850.8
625	2465.8	2506.9

Union Queries

RCs to RRDS or more disk drives to existing RCs. Component utilization figures, for the five IAT scenarios, are provided for each query type in tables 37 through 41. The utilization information is provided for each of the RRDS configurations modeled. Each table entry consists of four rows giving percent utilization for the four system components: Controller, Bus, RC subsystem, and Ms subsystem, respectively. Select queries, in Table 37, are Ms-intensive due to heavy data access and the requirement for address generation as well as reading records from the secondary storage. The Ms subsystem begins to saturate somewhere between the interarrival rates of 5 and 2.5 seconds, reaching 99.97 percent utilization when queries arrived, on the average, every 2.5 seconds. The utilization figures are lopsided, identifying the Ms as the only bottleneck, for Select queries. Controller and bus limitations are non-existent, even when the system is operating at 99.97 percent. Similarly, Table 38 reveals that the secondary storage creates the bottleneck when the arrival frequency of Insert queries approaches .063 seconds. This is due to the fact that the bulk of the work during Inserts concerns manipulation of the B+_tree directory. It is not

TABLE 37 COMPONENT PERCENT UTILIZATION FOR SELECT QUERIES

CONFIGURATION/ COMPONENT	SELECT IAT				
	40	20	10	5	2.5
2-RC Cont	.16	.30	.60	.77	.77
	.02	.05	.10	.12	.12
	1.08	2.17	4.26	5.34	5.35
	20.26	40.51	79.55	99.75	99.97
4-RC Cont	.16	.30	.62	1.23	1.52
	.02	.05	.10	.20	.25
	.55	1.11	2.22	4.36	5.42
	10.23	20.46	40.90	80.28	99.78
6-RC Cont	.16	.31	.63	1.25	2.21
	.02	.05	.10	.20	.36
	.38	.75	1.51	3.03	5.36
	6.88	13.76	27.51	54.99	97.37

TABLE 38. COMPONENT PERCENT UTILIZATION FOR INSERT QUERIES

INSERT IAT

CONFIGURATION/ COMPONENT		.50	.25	.125	.063	.031
2-RC	Cont	4.00	8.02	15.91	20.98	20.98
	Bus	.01	.02	.04	.05	.05
	RC	1.47	2.92	5.80	7.64	7.64
	Ms	18.95	37.75	74.92	98.76	98.76
4-RC	Cont	4.00	8.02	15.93	31.40	36.32
	Bus	.01	.02	.04	.08	.09
	RC	.83	1.65	3.29	6.48	7.50
	Ms	10.89	21.69	43.07	84.92	98.24
6-RC	Cont	4.00	8.07	16.11	32.11	44.59
	Bus	.01	.02	.04	.08	.12
	RC	.62	1.22	2.44	4.86	7.21
	Ms	8.28	16.55	33.06	65.90	99.67

TABLE 39. COMPONENT PERCENT UTILIZATION FOR JOIN QUERIES

JOIN IAT

CONFIGURATION/ COMPONENT		20000	10000	5000	2500	1250
2-RC	Cont	0.00	0.00	0.00	0.00	0.00
	Bus	.30	.60	1.21	2.32	2.39
	RC	12.60	25.20	50.40	96.83	99.66
	Ms	.09	.17	.35	.67	.69
4-RC	Cont	0.00	0.00	0.00	0.00	0.00
	Bus	.30	.60	1.20	2.42	4.62
	RC	6.30	12.60	25.20	50.40	96.43
	Ms	.04	.09	.17	.35	.67
6-RC	Cont	0.00	0.00	0.00	0.00	0.00
	Bus	.30	.60	1.20	2.41	4.80
	RC	4.20	8.40	16.80	33.60	67.20
	Ms	.03	.06	.11	.23	.47

TABLE 40. COMPONENT PERCENT UTILIZATION FOR DIFFERENCE QUERIES

CONFIGURATION/ COMPONENT		DIFFERENCE IAT				
		20000	10000	5000	2500	1250
2-RC	Cont	0.00	0.00	0.00	0.00	0.00
	Bus	0.00	0.00	0.00	0.01	0.01
	RC	12.60	25.20	50.40	97.01	99.63
	Ms	.09	.17	.35	.67	.69
4-RC	Cont	0.00	0.00	0.00	0.00	0.00
	Bus	0.00	0.00	0.00	0.01	0.02
	RC	6.30	12.60	25.20	50.40	97.01
	Ms	.04	.09	.17	.35	.67
6-RC	Cont	0.00	0.00	0.00	0.00	0.00
	Bus	0.00	0.00	0.00	0.01	0.02
	RC	4.20	8.40	16.80	33.60	67.20
	Ms	.03	.06	.11	.23	.47

TABLE 41. COMPONENT PERCENT UTILIZATION FOR UNION QUERIES

CONFIGURATION/ COMPONENT		UNION IAT				
		10000	5000	2500	1250	625
2-RC	Cont	2.52	5.00	10.08	19.42	19.97
	Bus	0.00	0.00	0.01	0.02	0.02
	RC	12.60	25.20	50.40	97.08	99.82
	Ms	.17	.35	.70	1.35	1.39
4-RC	Cont	2.52	5.04	10.08	20.16	40.32
	Bus	0.00	0.00	0.01	0.02	0.04
	RC	3.15	6.30	12.60	25.20	50.40
	Ms	.09	.17	.35	.70	1.40
6-RC	Cont	2.52	5.04	10.08	20.16	40.32
	Bus	0.00	0.00	0.01	0.02	0.04
	RC	1.40	2.80	5.60	11.20	22.40
	Ms	.06	.12	.23	.47	.93

surprising to see, in tables 39 through 41, that the RC subsystem creates the bottleneck for Join, Difference, and Union queries

because these two-relation query types are record-processing (CPU) intensive and require little directory management. Controller utilization for Union queries is higher than for Join and Difference queries because of the need to eliminate duplicate records by the controller, as results are received from the RCs. All five tables show roughly proportional decreases in bottleneck component utilization as RCs are added to the system (adding RC means adding both an RC and a dedicated disk drive). Most importantly, the two components where serial processing occurs, the controller and bus, were never factors in performance degradation for any query type, i.e., they were never the system bottleneck. This indicates that a feasible number of RCs (and associated disk drives) can be added to the configuration without overloading the controller or the bus.

Another goal of the RRDS performance analysis is to determine which operating environments are best suited for the system. An important operating environment characteristic is the user's query pattern. In Experiment 7 we were interested in determining the effects of various query patterns, on average RRDS response time, in order to draw conclusions about optimal operating environments. Different query patterns were simulated by running scenarios, containing all five query types, where the query mix was adjusted such that each scenario reflected a strong preponderance of a certain query type (i.e., a Select-intensive environment, an Insert-intensive environment, etc.). In addition, a scenario was run for an even mix of all the query types. Each

simulation run consisted of a total of 250 queries on the same database as in Experiment 6. Interarrival frequency was adjusted so that moderate queuing occurred but the system was not saturated. The experiment was performed on RRDS with 4-RCs, each RC having one disk drive.

The different query mix scenarios, as well as the results, are given in Table 42. The results indicate that RRDS performs best for the one-relation queries with the best operating environments being those which are Select or Insert-intensive. The poorest performance was observed for the Join-intensive

TABLE 42. QUERY MIX SCENARIOS AND RESULTS FOR EXPERIMENT 7

NUMBER OF QUERIES BY TYPE

SCENARIO	Select	Insert	Join	Difference	Union
1. Even Mix	50	50	50	50	50
2. Mostly Sel	150	25	25	25	25
3. Mostly Ins	25	150	25	25	25
4. Mostly Join	25	25	150	25	25
5. Mostly Diff	25	25	25	150	25
6. Mostly Uni	25	25	25	25	150

SCENARIO | AVERAGE RRDS RESPONSE TIME

1	803.17
2	422.01
3	396.47
4	1258.10
5	1128.60
6	746.93

scenario, due to its record processing overhead and inter-RC communication. The performance of the even query mix was encouraging, with an average response time falling halfway between that of the Select-intensive and Join-intensive environments, and not being skewed towards the poor performance of the two-relation scenarios. Based on these results it appears that RRDS will be best suited for applications featuring mostly one-relation operations. It should, however, be noted that the two-relation operations in RRDS are assumed to be on whole relations and not subsets of relations. The response time for the two-relation queries will improve if they operate on subsets of relations.

In the first six experiments of this performance analysis we varied certain parameters and observed system performance. In each case the percentage ideal goal was also calculated to see how well RRDS was achieving the design goal of performance proportional to the number of RCs. Experiment 8 has been designed specifically to observe extensibility and determine if this design goal has been met. In previous experiments when the number of RCs was increased the parameter under observation remained constant (e.g., in Experiment 2 we ran the three RRDS configurations on the same cardinality range). The approach in Experiment 8 is to increase both the size of the database and the number of RCs, in exact proportion, and see if response time remains constant. For each operation, 100 queries were run on three scenarios: a 2-RC RRDS with relations of 2500 records, a

4-RC RRDS with relations of 5000 records, and a 6-RC RRDS with relations of 7500 records. In each case relation schemas had 10 attributes with three directory/partitioning attributes. Results of this experiment, shown in Table 43, indicate that our design goal was achieved (or exceeded) in some cases and not in others. Performance of Select and Insert queries was excellent with Selects showing almost proportional performance. The serial processing steps in Select queries (parse and results transmission) were not a significant factor in performance. As RCs were added, and the database size increased, response time remained virtually the same. For Insert queries, adding RCs more than compensated for the increase in database size because the relation cardinality has little effect on the overall processing time. The results for the two-relation queries are not quite so encouraging. Joins exhibited degradation proportional to the increase in database size due to the serial inter-RC transmission of relation fragments via the bus. The transmission of larger partial results, to the controller following RC processing, also adversely affected performance. Similar results were observed for Difference queries for the same reasons. Though not as severe, response time degradation was also observed for Union queries due to the impact of the serial duplicate record elimination performed by the controller. These results support those of the previous experiment indicating strong RRDS performance and extensibility for retrieve and update-intensive environments and poorer performance when two-relation queries dominate the scenario.

TABLE 43. RESPONSE TIME AS NUMBER OF RCS AND RELATION CARDINALITY ARE INCREASED PROPORTIONALLY

RELATION CARDINALITY

CONFIGURATION	2500	5000	7500
2-RC	4.213		
4-RC		4.238	
6-RC			4.262

Select Queries

RELATION CARDINALITY

CONFIGURATION	2500	5000	7500
2-RC	.121		
4-RC		.080	
6-RC			.068

Insert Queries

RELATION CARDINALITY

CONFIGURATION	2500	5000	7500
2-RC	669.7		
4-RC		1364.0	
6-RC			2108.0

Join Queries

RELATION CARDINALITY

CONFIGURATION	2500	5000	7500
2-RC	661.8		
4-RC		1312.0	
6-RC			1968.0

Difference Queries

RELATION CARDINALITY

CONFIGURATION	2500	5000	7500
2-RC	400.5		
4-RC		596.4	
6-RC			923.1

Union Queries

At this point conclusions can be drawn concerning how well the RRDS design satisfies the goal of proportional performance. The combined results of the first five experiments, along with those of Experiment 8, indicate that proportional performance, to the number of RCs, can be expected when parameters affecting the parallel-processed portions of queries are altered. For example, data access is accomplished in parallel and the results of Experiments 4 and 5 indicated that the effect of increasing the number of predicates in queries, as well as varying predicate types, could be proportionally compensated by adding RCs. Conversely, increases in relation cardinality could not be proportionally offset for Join, Difference, and Union queries due to the fact that larger relation cardinalities resulted in larger message exchanges and more duplicate elimination, both serial actions. Increases in relation cardinality are shown to be proportionally offset for Select and Insert queries, by adding RCs, in the last experiment.

The final question to be answered, in the RRDS performance analysis, concerned the effect of adding more disk drives to the RCs in an Ms-intensive environment. Component utilization figures from Experiment 6 revealed that Select queries were Ms-intensive. In the final experiment the Select scenario from Experiment 6 was run with interarrival times set at average values of 5 and 2.5 seconds, respectively. We recall that these are the IAT ranges for which the 4-RC RRDS began to saturate in the Select scenario. Select queries were run on a 4-RC

simulation model and the number of disk drives per RC was varied from one to four. The records in a relation were assumed to be evenly distributed across the RCs. Furthermore, the portion of a relation in an RC was assumed to be evenly distributed across the disk drives for that RC. Results, given in Table 44, indicate proportional performance improvements up to the point where queuing at Ms diminishes. This indicates that Select response time degradation, due to query interarrival frequency, could be eliminated by adding disk drives to RRDS, a more cost-effective solution than adding RCs.

Conclusions

The RRDS performance analysis provided some interesting results relative to the design goals. First, we were interested in determining if, and when, proportional performance gains could be achieved by adding RCs. In addition, various operating environments and the effects of certain database, user, and query characteristics were explored. Finally, a set of predictive

TABLE 44. SELECT RESPONSE TIME AS DISK DRIVES ARE ADDED

SELECT IAT

DISK DRIVES/RC	5	2.5
1	17.62	100.30
2	4.87	10.77
4	4.02	4.40

performance figures was obtained, for a variety of scenarios, which can be verified by system prototypes.

The first five experiments revealed that RRDS performs best in scenarios which are retrieve and update intensive. The size of relations was found to have the most impact on query response time for the two-relation operations Join, Difference, and Union. Select queries were more affected by the types of predicates they contained than the number of predicates or the size of the target relation. The size of the query response set played a major role in the performance of Union queries due to the requirement for the controller to eliminate duplicate records. Percentage ideal goal figures revealed that the detrimental effects of increases in relation cardinality, and predicate characteristics, could be offset by adding RCs to the system. Increases in the size of the response set could be compensated to a lesser degree due to the increase in serial processing imposed on the controller.

The workload parameter most affecting RRDS performance was found to be query interarrival frequency. For each query type the point at which system throughput degraded was determined, as well as the component(s) responsible in each case. In all cases, the component(s) identified as bottlenecks can be replicated to offset the effects of interarrival frequency, indicating a strong degree of system extensibility. These results also indicated the absence of the controller limitation problem, one of the design goals outlined in Chapter 4.

The operating environment for which RRDS performed the worst was found to be the one with a preponderance of Join queries. The optimal environments for the system are those with mostly Select and Insert queries. The fact that scenarios with even mixes of query types were not more affected by the poor performance of the two-relation queries was especially encouraging. A cost-effective alternative to adding RCs to compensate for performance degradation, in the Select-intensive environment, is to add disk drives to existing RCs. In an experiment to determine if constant response times were attainable as relation cardinality and number of RCs were increased proportionally, Select and Insert query response times remained constant or actually improved. On the other hand, the two-relation query response times degraded, indicating that the RRDS design's main weakness lies in its processing of two-relation queries, particularly Joins.

The design and performance analysis of RRDS is now complete. In the next chapter we will look at RRDS performance relative to three other software-oriented, multi-backend, database systems.

CHAPTER 9

A PERFORMANCE EVALUATION OF RRDS WITH RESPECT TO THREE OTHER MULTI-BACKEND DATABASE SYSTEMS

In this chapter we evaluate the RRDS architecture with respect to three other relational multi-backend systems--DIRECT (DeWitt 1979), RDBM (Auer 1980), and MDBM (Srinidhi 1982). DIRECT and RDBM were discussed in Chapters 2 and 4. MDBM, a system based upon magnetic bubble memory technology, is chosen for comparison because of the similarity of its physical architecture to that of RRDS. This system, however, employs different data access, data placement, and directory management strategies. For the evaluation we consider an MDBM with conventional disk drives substituted for the magnetic bubble memories.

For each of these relational systems three types of queries (Selection, Join, and aggregate function) are studied. The motivation for query response time comparisons comes from a paper by Dewitt and Hawthorn (1980). We will use the same hardware parametric model and analytical modeling approach. This chapter contains a brief discussion of the architectures modeled as well as the hardware parametric models for the disk, processor, and communication subsystems. It also provides the analytical models and performance comparison results for the three query types on the four systems.

The Architectures Used In The Comparison Study

Of the three architectures selected for analysis DIRECT and RDBM have already been discussed in some detail (chapters 2 and 4). According to the classification in DeWitt (1980) both architectures belong to the multi-processor cache (MPC) category of database systems. They feature a set of general-purpose processors and a three-level memory hierarchy. The top level of the memory hierarchy consists of the internal memories of the processors, assumed to be large enough to accommodate a compiled query and at least one block of data. The middle level of the hierarchy consists of a disk cache which is assumed to hold a block of data for each processor. In DIRECT this level is made up of a set of charge-coupled device (CCD) modules, and, in RDBM it consists of a page buffer. Finally, the bottom level of the memory hierarchy is made up of mass storage devices (disk drives) which hold the database. The unit of transfer between memory hierarchy levels is a block of a relation. The mass store and disk cache communicate over a bus. The interconnection between the processors and disk cache permits each processor to simultaneously access its block of the disk cache, and, allows all the processors to simultaneously read the same block of cache.

For this analysis the major difference between DIRECT and RDBM is in the way they handle two-relation queries, such as Joins. DIRECT uses its parallel processors for all relational operations including Joins. RDBM, on the other hand, is an

architecture based upon functional specialization. This system utilizes a dedicated Join processor which accesses the database through the memory hierarchy.

MDBM (Srinidhi 1982) partitions the relations of the database across a set of mass storage devices which are directly accessed by a set of general-purpose processors, communicating with a controller over a bus. The MDBM architecture is illustrated in Figure 62. Components of interest are the DAMP, C.Bus, ICs, and secondary storage devices. The data managing processor (DAMP), analogous to the controller in previously discussed systems, provides the interface between the intelligent controllers (ICs) and the host processor. The DAMP accomplishes data allocation among the areas under control of the different ICs as well as distribution of the subqueries to the various ICs. The C.Bus is used for communication between the DAMP and the ICs. The unit of transfer is a block of information consisting of several pages of magnetic bubble memory (MBM) units with the same logical address. A bus arbitrator unit processes demands for the bus in round-robin fashion, where requests are entered into a first-in first-out (FIFO) queue. For the purpose of this evaluation we assume that the C.Bus provides broadcast capability from the DAMP to the ICs. The intelligent controllers (ICs), and their associated components (controller memory (CMEM), communications processor (CP), and data buffer area (DB)), form the heart of MDBM. The ICs are responsible for handling their regions of the data area and all query processing associated with

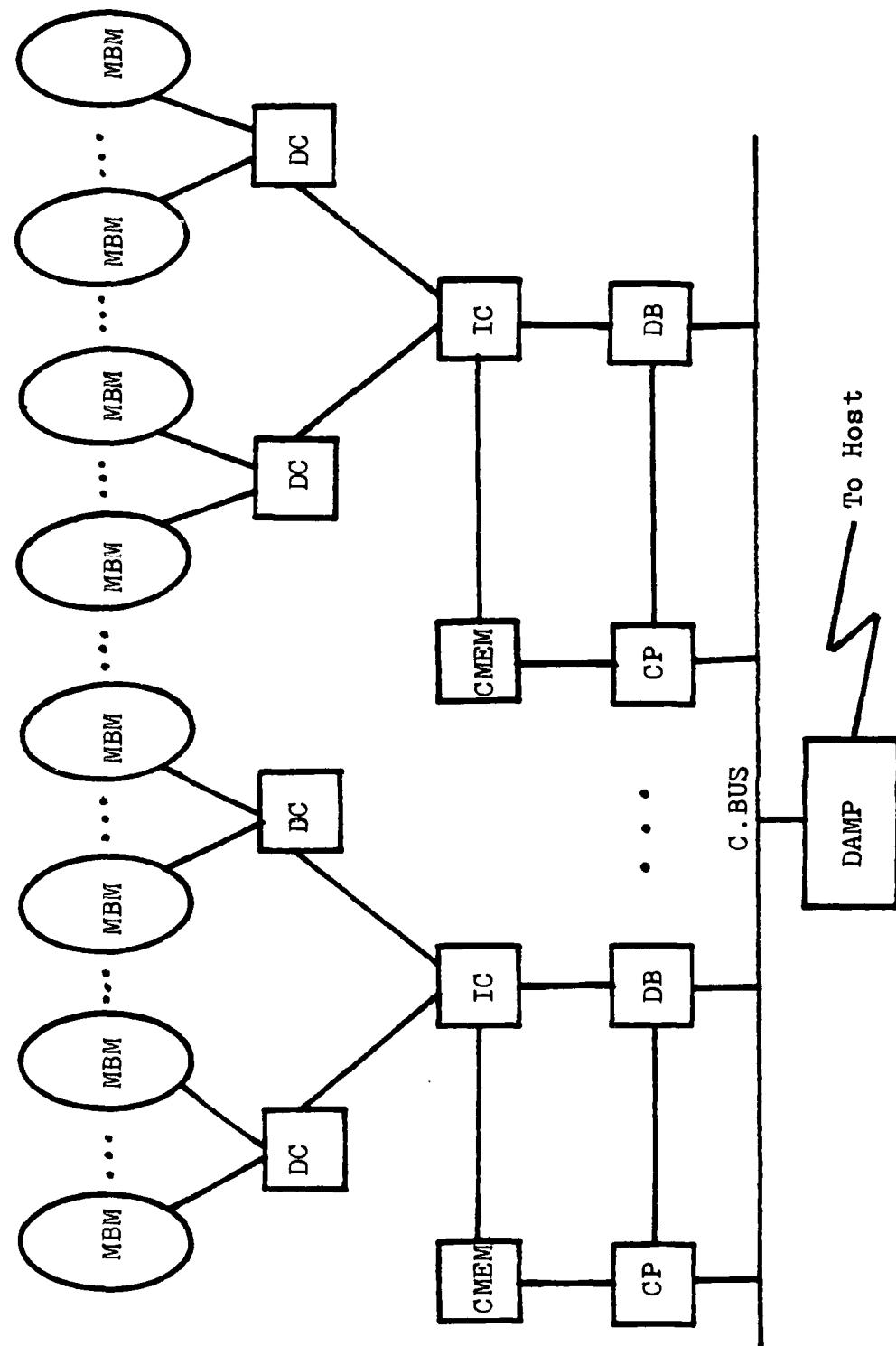


Figure 62. MDBM Architecture

this data area. The final components are the secondary storage devices. MDBM was designed specifically to exploit the advantages of the MBM technology emerging in the early 1980s. Unfortunately, economic factors affected the sales of MBM units and research/development in this area has tapered. For this reason, in our analysis, we assume conventional moving-head disk units for the mass storage devices in MDBM.

The Parametric Model

The hardware parametric model is identical to that used in DeWitt (1980). The mass storage device, used in all four architecture models, is assumed to be an IBM 3330 disk drive with parameters and values provided in Table 45. All processors are assumed to be 1-MIP units, such as a VAX 11/780, with parameters presented in Table 46. The parameter Tsc represents the time to perform a simple query operation (such as relation scanning in Select operations) on a block of data. Tblk corresponds to the CPU time to accomplish a complex query, such as a Join, on a

TABLE 45. IBM 3330 DISK PARAMETERS

PARAMETER	DESCRIPTION	VALUE
b	Block size	13030 bytes
DCYL	Blocks per cylinder	19
Tio	Block read/write time	16.7 msec
Tdac	Average access time	38.6 msec
Tsk	Track-track seek time	10.1 msec

block of data. Derivation of these values is explained fully in (DeWitt 1980).

In DeWitt (1980) and Srinidhi (1982) the MPC architectures and MDBM are modeled as a controller and nineteen processors (the disk has nineteen cylinders). The MPC model features a cache of nineteen blocks of RAM and one standard IBM 3330 disk drive. In order to examine the contents of a block of the disk it must first be moved to the disk cache and then to the local memory of each processor. In MDBM and RRDS each backend processor reads blocks of data directly from its dedicated IBM 3330 disk drives. In our analysis we added another parameter, #PU, indicating the number of parallel processors (processing units) in each system. Results, based on variance of this parameter, provided information on the effect of parallelism for each architecture.

In the MPC systems (DIRECT and RDBM) the backend processors are connected to the controller by an output channel operating

TABLE 46. PROCESSOR PARAMETERS

PARAMETER	DESCRIPTION	VALUE
Tsc	CPU time to perform simple query operation on block	10.0 msec
Tblk	CPU Time to perform complex query operation on block	95.0 msec
Toio	CPU time to initiate I/O operation	2.0 msec
Tcodegen	CPU time to compile a query	152.0 msec
Tmsg	CPU time to send/receive a message	2.0 msec

asynchronously and independently from the processors. The bandwidth of the channel, assumed in DeWitt (1980), is 2 Mbytes/second, corresponding to that of the VAX 11/780's Mass Bus adapter. Based on this bandwidth figure, the transfer time for one block of 13,030 bytes, T_{bt} , is 6.5 msec. In both the MDBM and RRDS models the busses, as well as the links between processors and disk drives, are also assumed to have bandwidths of 2 Mbytes/second. The 2 Mbytes/second bus bandwidth is compatible with that of the output channel, ensuring result tuples are not formed faster than they can be transferred.

In the next section, given the four comparable models, we benchmark each database system on Select, Join, and aggregate function operations. These operations were chosen because each represents a class of query types. Select queries represent operations which can be performed in $O(n)$ time, for n tuples, on a single processor. Join queries are representative of relational operations involving two relations and requiring either $O(n \log n)$ or $O(n^2)$ time on a single processor. Aggregate functions serve as a benchmark for complex operations on single relations, requiring $O(n \log n)$ time on a single processor, such as Projection with duplicate elimination.

Performance Comparisons

The performance evaluations, presented in this section, measure the total system work necessary to process a query and not the query response time. The expressions represent query

processing from compilation, by the respective system controller, to receipt of results, from the backend network, by the controller. Transmission of results from the controller to the host is not included. The number of disk blocks occupied by a relation R is denoted by $|R|$, and the selectivity factors for Select and Join queries are denoted by f and jsf , respectively. In the following analysis, the IBM 3330 parameters have been substituted for the MBM parameters in the MDBM query processing expressions derived in Srinidhi (1982).

Selection Queries

The timing expressions for the MPC (DIRECT and RDBM) and MDBM designs have been derived in DeWitt (1980) and Srinidhi (1982) and are repeated here. In the expressions it is assumed there is no index on the attribute being qualified. If such an index exists then the expressions $|R|/19$ for the number of cylinders searched and $|R|$ for the number of blocks of R to be processed are assumed to be replaced by $(|R|*f)/19$ and $|R|*f$, respectively.

$$\begin{aligned} T(\text{Select-DIRECT/RDBM}) = & \quad T\text{codegen} + T\text{msg} + T\text{dac} + \\ & \max\{([|R|/\text{DCYL}] - 1)*T\text{sk} + |R|*T\text{io} + (\text{DCYL}/\#\text{PU})(T\text{io} + T\text{sc}), \\ & ([|R|*f] - 1)(T\text{msg} + T\text{bt})\} + T\text{msg} + T\text{bt} \end{aligned}$$

```

T(Select-MDBM) = Tcodegen + Tmsg + Tdac +
max{(|R|/#PU) - 1)*Tdac + [|R|/#PU](Tio + Tsc),
(|R|*f) - 1)(Tmsg + Tbt)} + Tmsg + Tbt

```

In the RRDS architecture the query is compiled in the controller and broadcast to the RCs maintaining equal portions of the database. This requires (Tcodegen + Tmsg) msec accounting for compilation and broadcast of the query to the RCs. After receiving the Select query, the RCs perform processing according to the Select algorithm in Chapter 8. Each RC processes an equal-size portion of the relation, due to the RRDS data placement strategy. The portion of R processed by each RC, in parallel, is ($|R|/#PU$) and the processing time consists of accessing the disk, reading the blocks of data and selecting qualifying tuples, expressed by (Tdac + Tio + Tsc). The output blocks, $|R|*f$ blocks, are transferred to the controller via the bus. This process can be performed in parallel with the query execution except for the transmission of the last block of result data. Hence, the query execution time for RRDS is given by:

```

T(Select-RRDS) = Tcodegen + Tmsg +
max{(|R|/#PU)(Tdac + Tio + Tsc), [(|R|*f) - 1](Tmsg + Tbt)} +
Tmsg + Tbt

```

The results of evaluating these formulas are given in tables 47 and 48, for no-index and index cases, for a representative relation of 50,000 tuples, with each tuple being a fixed-length 100 bytes. The values given are for selectivity factors ranging from .0001 to .1. From Table 47 we observe that RRDS and MDBM perform approximately five times better than the MPC organizations DIRECT and RDBM. MDBM and RRDS perform about the same for the no-index case. The performance improvement of these two systems is because of the parallel reading of relation fragments from the disks. In DIRECT and RDBM loading the disk

TABLE 47. EXECUTION TIME FOR SELECTION ON R WITH NO INDEX

SELECTIVITY FACTOR	DIRECT	RDBM	MDBM	RRDS
.0001	6.818	6.818	1.483	1.481
.001	6.818	6.818	1.483	1.481
.01	6.818	6.818	1.483	1.481
.1	6.818	6.818	1.483	1.481

TABLE 48. EXECUTION TIME FOR SELECTION ON R WITH INDEX

SELECTIVITY FACTOR	DIRECT	RDBM	MDBM	RRDS
.0001	.312	.312	.266	.227
.001	.312	.312	.266	.227
.01	.362	.362	.266	.227
.1	.965	.965	.572	.595

cache, from the disk, is a serial operation causing poor system performance. Only the I/O from the disk cache and the Select processing are accomplished in parallel in these MPC organizations. For all values of f the time to read data from the secondary storage and performing the Select operation on R dominated the time to transmit results. For the case where there was an index on the qualifying attribute the performance for all systems was similar except for the largest selectivity factor value (.1). At this point more than one block of data was being read from the secondary storage and the advantages of reading the data in parallel, in MDBM and RRDS, again resulted in reduced execution times. In all the designs considerable performance improvements are achieved when an index exists on the qualifying attribute.

In tables 49 and 50 results of varying the number of processors for a selection factor of .1, for the no-index and index cases, respectively, are presented. For the MPC architectures the degree of parallelism had little effect on Select execution times as the number of processor/cache units were increased from 5 to 30. This is because, for DIRECT and RDBM, the execution time is dominated by the serial reading of blocks of R from the disk into the cache areas. Adding processors improved the performance of RRDS and MDBM as the amount of work accomplished by each processor was reduced. As before, disk I/O and query execution time dominated results output time, even for the 30 processor configuration. The

TABLE 49. EXECUTION TIME FOR SELECTION ON R WITH NO INDEX AND VARIABLE NUMBER OF PROCESSING UNITS

#PU	DIRECT	RDBM	MDBM	RRDS
5	6.904	6.904	5.150	5.154
10	6.853	6.853	2.668	2.668
15	6.836	6.836	1.833	1.833
20	6.829	6.829	1.415	1.415
25	6.829	6.829	1.164	1.164
30	6.829	6.829	.997	.997

TABLE 50. EXECUTION TIME FOR SELECTION ON R WITH INDEX AND VARIABLE NUMBER OF PROCESSING UNITS

#PU	DIRECT	RDBM	MDBM	RRDS
5	.965	.965	.663	.663
10	.914	.914	.527	.489
15	.896	.896	.527	.489
20	.890	.890	.527	.489
25	.890	.890	.527	.489
30	.890	.890	.527	.489

results, for MDBM and RRDS, are quite different for the index case. For ten or more processors, when there was an index on the qualifying attribute, the amount of processing required by each processor was so small that the serial transmission of results dominated the performance and negated the benefit of adding more parallel processors. Despite losing the advantage of parallelism, for cases with more than ten processors, MDBM and RRDS still outperformed the MPC architectures.

Join Queries

If R and S are two relations to be joined, we use $|R|$ and $|S|$ to denote the number of blocks occupied by the relations, respectively. The join selectivity factor is denoted by jsf . The DIRECT database system employs the MPC approach to perform Join operations. The timing expression for the MPC approach, a block-parallel version of the nested loops algorithm, has been derived in DeWitt (1980) and is shown below:

$$T(\text{Join-DIRECT}) = T\text{codegen} + T\text{msg} + \max\{T\text{join}, T\text{send_results}\} + T\text{msg} + T\text{bt}$$

Where:

$$T\text{join} = (|R|/\#PU)\{T\text{dac} + \#PU*T\text{io} + T\text{io} + T\text{dac} + [(|S|/19) - 1]*T\text{sk} + 2*T\text{io} + |S|*T\text{blk}\}$$

And

$$T\text{send_results} = [(|R|*|S|*jsf) - 1] * (T\text{msg} + T\text{bt})$$

RDBM, which utilizes an MPC approach for Select operations, has a dedicated processor, called interrecord processor (IRP), for accomplishing two-relation operations such as Joins. The join processor is assumed to have the same characteristics as the 1-MIP processor previously discussed and, as such, accesses the blocks of R and S through the three-level memory hierarchy. For the purpose of this analysis we assume the RDBM join processor uses an optimized conventional single-processor join algorithm

based upon the sort-merge approach. The timing expression for conventional system Join operations is derived in (DeWitt 1980) and is presented below:

$$T(\text{Join-RDBM}) = T\text{codegen} + T\text{msg} + T\text{sort_R} + T\text{sort_S} + T\text{merge}$$

Where $T\text{sort_X}$ for a relation of X blocks is given by:

$$T\text{sort_X} = (\log X \text{ base } 4) * 2 * \{T\text{dac} + [(X/19) - 1] * T\text{sk} + X(T\text{io} + T\text{oi})\} + (\log X \text{ base } 4) * [X * (T\text{blk}/2)]$$

And

$$T\text{merge} = (|R| + |S|) * \{T\text{dac} + T\text{io} + T\text{oi}\} + (|R| + |S|) * T\text{blk}/2$$

The MDBM Join algorithm, used in developing the timing expression derived in Srinidhi (1982), is similar to the RRDS algorithm. The timing expression for MDBM Join operations is presented below:

$$T(\text{Join-MDBM}) = T\text{codegen} + T\text{msg} + T\text{dac} + T\text{io} + T\text{bt} + \max\{T\text{join}, T\text{send_results}\} + T\text{msg} + T\text{bt}$$

Where

$$T\text{join} = \max\{(|S| - 1) * (T\text{io} + T\text{dac} + T\text{bt}), (|R|/\#PU) * (T\text{dac} + T\text{io} + T\text{blk}) * |S|\}$$

And

$$T\text{send_results} = [(|R| * |S| * jsf) - 1] * (T\text{msg} + T\text{bt})$$

As described in Chapter 8, Join operations in RRDS require transfer of the smaller relation (involved in the operation) among the RCs. Let us assume $|S|$ is smaller than $|R|$. The Join operation in RRDS proceeds according to the algorithm presented in Chapter 8. The algorithm requires an exhaustive search of all blocks for relation S, and hence represents a worst-case situation for Join processing. The time to compile the query and forward it to the RCs requires $(T_{codegen} + T_{msg})$ msec, as in Select operations. Each RC broadcasts its blocks of the smaller relation ($|S|$) sequentially, over the bus, to the other RCs. For each block received, the RCs scan through the blocks of their portion of relation R performing the Join operation. Each operation between a block of relation R and a block of relation S requires T_{blk} milliseconds. The resulting output blocks are sent, over the bus, to the controller. Assuming, as in Srinidhi (1982), that transfer of results to the controller (except the last block), and transfer of the blocks of S among the RCs, can proceed in parallel with the join processing, we obtain the following timing expression:

$$T(\text{Join-RRDS}) = T_{codegen} + T_{msg} + \max\{T_{join}, T_{send_results}\} + T_{msg} + T_{bt}$$

Where:

$$T_{join} = \max\{|S| * (T_{dac} + T_{io} + T_{bt}), \\ (|R|/\#PU) * (T_{dac} + T_{io} + T_{blk}) * |S|\}$$

And

```
Tsend_results = [(|R| * |S| * jsf) - 1] * (Tmsg + Tbt)
```

The results of evaluating these formulas are given in tables 51 and 52, for a join operation on relations with no index on the joining attributes. The cardinalities of relations R and S are 10,000 tuples of 100 bytes each and 3000 tuples of 75 bytes each, respectively. Table 51 gives the results for join selectivity factors ranging from .0001 to .1. The join selectivity factor had no effect on performance as the time to accomplish join processing overwhelmed the time to transmit result blocks, even for the largest value of jsf (.1). Of the four systems, RDBM performed the worst because of its conventional algorithm running on a single processor. Recall, however, that RDBM is based on functional specialization. Providing a dedicated Join processor frees the parallel processors for simultaneously accomplishing other operations, a factor not included in this analysis. The MPC organization of DIRECT slightly outperformed both MDBM and RRDS. This is because serial transfer of relation blocks is not required in DIRECT. As expected, RRDS and MDBM performed approximately the same since they are based on the same algorithm and similar organization.

The results depicted in Table 52 are for joins on R and S with a join selectivity factor of .1. The number of processing units, in each system, was varied from 5 to 30. The number of parallel processors had no effect on RDBM Join performance

TABLE 51. EXECUTION TIME FOR JOINS ON R AND S

SELECTIVITY FACTOR	DIRECT	RDBM	MDBM	RRDS
.0001	7.7	34.0	10.7	10.6
.001	7.7	34.0	10.7	10.6
.01	7.7	34.0	10.7	10.6
.1	7.7	34.0	10.7	10.6

TABLE 52. EXECUTION TIME FOR JOINS ON R AND S FOR VARIABLE NUMBER OF PROCESSING UNITS AND FIXED JOIN SELECTIVITY FACTOR

#PU	DIRECT	RDBM	MDBM	RRDS
5	26.84	34.00	40.04	39.98
10	14.91	34.00	20.12	20.06
15	10.52	34.00	13.48	13.42
20	8.25	34.00	10.16	10.09
25	6.89	34.00	8.18	8.13
30	5.99	34.00	6.86	6.80

because this system utilizes a dedicated processor for two-relation operations. Performance of DIRECT, MDBM and RRDS improved as processors were added since the dominating join effort on each processor was reduced. As before DIRECT outperformed MDBM and RRDS over the entire range of #PU, however, for low values of #PU the performance of MDBM and RRDS was almost 50 percent worse than for DIRECT. As the number of parallel processors in MDBM and RRDS was decreased the size of the relation fragments transferred, prior to joining, increased as well as the number of blocks processed by each IC/RC (the most

costly part of the Join operation for these architectures). On the other hand, adding more processors had no effect on the expensive serial transfer of blocks of R and S from the disk to the cache in DIRECT. The fact that adding processors to MDBM and RRDS yields almost proportional performance gains is significant and it is important to note that, as the number of processors is increased, the gap between the performance of DIRECT and the MDBM/RRDS approach narrows.

Aggregate Function Queries

The final relational operation evaluated is the aggregate function. In DeWitt (1980) the importance of aggregate functions, as a benchmark for complex one-relation operations, is explained. There are two basic types of aggregate queries supported by database systems: scalar aggregates and aggregate functions. Scalar aggregates are aggregations over an entire relation, such as those discussed in Chapter 8. An example of a scalar aggregate query is given below:

Query: SUM (SALARY) From EMPLOYEES

Result: 249,000

Aggregate functions, on the other hand, first divide the relation into non-intersecting partitions (based on some attribute, e.g., DEPT) and then compute scalar aggregates on the individual partitions. Thus, given a target relation an aggregate function produces a set of results (i.e., a result relation). An example

of an aggregate function is given below:

Query: SUM (SALARY) From EMPLOYEES By DEPT

Result:

DEPT	SALARY
Pet	139000
Toy	110000

We now develop timing expressions for the four database system architectures being studied.

DIRECT and RDBM both execute aggregate functions according to the MPC approach given in DeWitt (1980). Each processor reads a set of source relation blocks and computes an aggregate value for each partition in R. It is assumed that the partitions are evenly distributed throughout the relation, i.e., each processing unit sees every partition and calculates aggregate values on each. The processors maintain the relation partitions in sorted order and, as each new block of R arrives at a processor, each tuple in the block is placed in the correct partition (located by binary search) and the partition aggregate value is updated. The cost of processing the tuples in a block is assumed to be T_{blk} . Calculation of partition aggregate values in the processors produces a set of partial result blocks, denoted by $|N|$. The partial result blocks are then written back to the disk. In the second step a parallel-merge algorithm is used to combine the partial results for transmission to the controller. The MPC timing expression for aggregate functions is taken from DeWitt

(1980) and shown below:

$$T(\text{Agg-DIRECT/RDBM}) = T\text{codegen} + T\text{msg} + T\text{process_R} + T\text{prior} + T\text{parallel_merge} + T\text{return_results}$$

Where:

$$T\text{process_R} = T\text{dac} + (|R|/\#PU) * [T\text{sk} + (\#PU * T\text{io}) + T\text{io} + T\text{blk}]$$

$$T\text{prior} = T\text{dac} + (|N| - 1) * T\text{sk} + (|N| * \#PU) * T\text{io}$$

$$T\text{parallel_merge} = 2 * (\log N \text{ base } 2) * \{T\text{dac} + (|N| - 1) * T\text{sk} + |N| * [(\#PU * T\text{io}) + T\text{io}] + [(\log |N| \text{ base } 2) * |N| * T\text{blk}] + T\text{dac} + (|N| - 1) * T\text{sk} + (|N| * 19 * T\text{io}) + [|N| * (\log \#PU \text{ base } 2)] * (2 * T\text{io} + T\text{blk} + T\text{io})\}$$

And

$$T\text{return_results} = |N| * (T\text{msg} + T\text{bt})$$

RRDS is designed to employ only conventional components. In our analysis we assume RRDS consists of conventional 1-MIP processors, each with dedicated IBM 3330 disk drives. The database is distributed across the processors, according to the data placement strategy, described in Chapter 6, such that each

processor contains an equal portion of each relation in the database. Furthermore, we assume that the portion of a relation, located at each RC, contains an even distribution of tuples in the relation's aggregate function partitions. After receipt of a compiled query from the controller, requiring $(T_{codegen} + T_{msg})$ msec, aggregate function processing in RRDS proceeds in three steps. First, each RC calculates the aggregate function for its portion of the relation R . Next, the RCs transmit these partial results, via the bus, to the controller. After receiving all the partial results the controller performs a final aggregate function calculation, on the blocks of partial results, producing the final result.

The first step is accomplished, in parallel, by the RCs. We assume the RCs use DeWitt's conventional single-processor algorithm, based upon a four-way merge sort. At each RC, the relation is sorted on the partitioning attribute in order to bring all the tuples in the same partition together. The size of the relation fragment sorted by each RC is $(|R|/\#PU)$ blocks. The sort algorithm is an optimized four-way merge-sort DeWitt (1980). In the last phase, instead of completing the sort, the aggregate value for each partition is computed. As before, computation of an aggregate value for each partition is assumed to be a complex operation with each block requiring T_{blk} msec. The time needed to accomplish this partial result processing is denoted by T_{ppr} . The output of this step is an aggregate value, for each partition, at each RC.

In the second step, the $|N|$ blocks of partial results, computed by each RC, are sent over the bus serially to the RRDS controller. Each RC will send $|N|$ blocks of data at a transmission cost of T_{bt} per block. The time needed to send the partial result blocks is denoted by T_{spr} .

Computing the final results, designated T_{pfr} , utilizes the same merge-sort algorithm as the first step. This processing is performed on the partial results received from the RCs, however, and there is no disk I/O. Results received by the controller for processing consist of $(\#PU * |N|)$ blocks.

The RRDS timing expression for computing aggregate functions is now given:

$$T(\text{Agg-RRDS}) = T_{codegen} + T_{msg} + T_{ppr} + T_{spr} + T_{pfr}$$

Where:

$$\begin{aligned} T_{ppr} = & \{ [2 * (\log (|R|/\#PU) \text{ base } 4)] - 1 \} * \\ & \{ T_{dac} + [((|R|/\#PU)/19 - 1) * T_{sk}] + \\ & (|R|/\#PU) * (T_{oi} + T_{io}) \} + \\ & (\log (|R|/\#PU) \text{ base } 4) * [((|R|/\#PU)/2] * T_{blk} + \\ & (|R|/\#PU) * T_{blk} \end{aligned}$$

$$T_{spr} = \#PU * (|N| * T_{bt})$$

$$\begin{aligned} T_{pfr} = & [\log (\#PU * |N|) \text{ base } 4] * [(\#PU * |N|)/2] * T_{blk} + \\ & (|N| * \#PU * T_{blk}) \end{aligned}$$

MDBM, due to its architectural similarity to RRDS, could certainly employ the same aggregate function algorithm. However, in order to explore an alternative, we provide a different method for calculating the final results. Since calculation of an aggregate function on the partial results could cause the controller to become a system bottleneck, we consider, in MDBM, the case where final results are calculated by the IC units in cascade.

After compilation and transmission of the query, requiring $(T_{codegen} + T_{msg})$ msec, the first step of the MDBM algorithm is identical to that of the RRDS approach, requiring T_{ppr} . The ICs, in parallel, apply the DeWitt (1980) conventional single-processor algorithm to their respective portions of R . The portion of R operated on by each IC is again $(|R|/\#PU)$. After this step, each IC contains $|N|$ blocks of partial results. Next, the first IC transmits its partial result blocks to the adjacent IC, where the same conventional single-processor algorithm is applied to the $2*|N|$ blocks of partial results. Following calculation of a new partial result ($|N|$ blocks) at IC#2 the result is transmitted to the next IC, and the process is repeated. This transmission-calculation sequence continues until the last IC performs the aggregate function on its partial result (from step 1) and the partial result received from the adjacent IC. There will be a total of $(\#PU - 1)$ such cycles where each consists of: 1) transmission of $|N|$ blocks of partial result data at a cost of T_{bt} msec each, and 2) calculation of the

aggregate function on the $2*|N|$ blocks of partial result data. The transmission phase of the cascade is denoted by $T_{transmission}$ and the aggregate function calculation is denoted by T_{ca} (time to calculate answer).

The final component of the MDBM timing expression, denoted by T_{send_result} , represents the time needed to transmit the final result, calculated by the last IC, to the DAMP. This requires $(|N|*Tbt)$ msec. The timing expression for aggregate functions in MDBM is given below:

$$T(\text{Agg-MDBM}) = T_{codegen} + T_{msg} + T_{ppr} + T_{transmission} + [(\#PU - 1) * T_{ca}] + T_{send_result}$$

Where:

$$T_{transmission} = (\#PU - 1) * (|N|) * Tbt$$

$$T_{ca} = [(\log (2*|N|) \text{ base } 4) * |N| * Tblk] + [2*(|N|) * Tblk]$$

$$T_{send_result} = |N| * Tbt$$

The aggregate function timing expressions were evaluated for a variable number of partitions of R , denoted by P , ranging from 5 to 5000. Relation size was 50,000 tuples of 100 bytes each, and each system initially consisted of nineteen parallel processors. Results, presented in Table 53, indicate that the RRDS and MDBM approaches perform similarly for between 5 and 250 partitions of R . This is because, when there are relatively few

TABLE 53. EXECUTION TIME FOR AGGREGATE FUNCTION QUERIES
FOR VARIABLE NUMBER OF PARTITIONS OF R

PARTITIONS	DIRECT	RDBM	MDBM	RRDS
5	10.2	10.2	5.7	5.6
50	10.2	10.2	7.5	6.7
250	10.3	10.3	15.6	13.7
500	19.3	19.3	28.8	23.5
2500	89.9	89.9	159.7	116.1
5000	193.2	193.2	347.2	244.2

partitions (i.e., $|N|$ is small), the serial actions of transmitting partial results and single-processor calculation of final results have little effect compared to the time needed to perform disk I/O and calculate partial results on $|R|$. For the low value range of P , both MDBM and RRDS outperform the MPC organizations of DIRECT and RDBM. As the number of partitions increases above 250, however, the serial partial results transmission and single-processor final result calculation of RRDS and MDBM take their toll on performance. As the serial disk I/O, required to load the cache of the MPC systems, has less effect on overall time (when P is large) the performance of DIRECT and RDBM slowly overtake that of MDBM and RRDS. The cascaded final result calculation of our algorithm for MDBM became more of a performance liability as the number of partitions increased over 250, and the response time values of RRDS and MDBM began to diverge at this point. The fact that MDBM and RRDS perform well for values of P up to 250 is encouraging since, in practical applications, relatively few partitions are

more likely. For example, if the aggregate function was computed on the attribute DEPT, a P-value of 5000 would mean the relation contains 5000 different values for DEPT. In other words, if R contains 10,000 tuples and P is 5000, each partition will contain only two tuples. Viewed in this light, the superior performance of the MPC machines, for high values of P, has less significance. Conversely, the fact that RRDS and MDBM outperform the MPC machines almost two-to-one, when P is 5 to 50, is more encouraging.

The final result, presented in Table 54, is for a variable number of processing units and a fixed value for P. The value 250 was chosen for P since it was determined to be the performance threshold value in the previous experiment. The performance of all four systems improved as processors were added, up to a point. Then, as #PU increased, the performance of all four systems began to degrade. The optimal number of

TABLE 54. EXECUTION TIME FOR AGGREGATE FUNCTION QUERIES FOR VARIABLE NUMBER OF PROCESSING UNITS

#PU	DIRECT	RDBM	MDBM	RRDS
5	20.1	20.1	29.1	28.5
10	13.6	13.6	13.7	12.5
15	12.9	12.9	13.3	11.7
20	12.7	12.7	15.9	13.9
25	12.8	12.8	17.4	15.2
30	13.1	13.1	19.5	17.1

processors appears to be 15 to 20, depending on the system being observed. When #PU was in the range of 10 to 15, RRDS actually performed better than the MPC organizations for 250 partitions. Again, the performance of RRDS and MDBM was very close, with RRDS performing slightly better. The performance anomaly, exhibited for MDBM and RRDS as #PU increased above 15, is due to the increase in partial results transmission and the attendant decrease in aggregate function calculation effort per processor. Similarly, the performance degradation for the MPC architectures is due to the increased volume of partial results transmission and the added disk I/O as they are written back to the disk.

This concludes the comparison of RRDS with DIRECT, RDBM, and MDBM in terms of query processing times. RRDS and MDBM were found to perform almost the same, when conventional disk drives were substituted for the magnetic bubble memory units of MDBM. It should be noted, however, that these timing expressions do not incorporate the details of such software design features as data placement, directory management, and data access. Both RRDS and MDBM performed better, for Select and aggregate function queries, than the MPC approaches of DIRECT and RDBM. DIRECT performed the best for Join operations with RRDS and MDBM close behind. RDBM, using a single processor for join operations, exhibited the worst performance. Adding more processors to RRDS and MDBM improved the performance of Join queries, on these systems, to a point close to that of DIRECT. In addition, worst case algorithms were assumed for RRDS and MDBM for this analysis. Optimizing the join

algorithms could improve the performance of these two systems. The performance of RRDS and MDBM was encouraging for the aggregate function queries for low number of partitions, the most likely aggregate queries. Finally, a performance anomaly was observed for aggregate function queries in all the architectures: as processors were increased beyond a certain number, the system performance degraded (the serial communication overhead caused the degradation).

In the next chapter we present conclusions and suggestions for future research.

CHAPTER 10

SUMMARY, CONCLUSIONS, AND SUGGESTIONS FOR FUTURE RESEARCH

The goal of this research was to design a relational multi-backend database system to explore the possibility of using multiple, commercially-available minicomputers and disk drives to achieve performance improvements and capacity growth. We now summarize the work, present conclusions and offer suggestions for further research.

Summary

The first step in designing RRDS was to develop a set of design goals for such a system. These goals were determined by surveying existing software-oriented multi-backend database systems and studying their advantages and limitations. First, RRDS had to support the relational data model with a complete DML and query processing capability for both one and two-relation operations. In order to be feasible RRDS, based on a multi-processor architecture, must make maximum use of parallel processing for improved throughput, exhibit performance inversely proportional to the number of processors, and be extensible. Finally, RRDS must not contain any of the custom components characteristic of other database machines. Instead, the design would be based only on off-the-shelf components. Based on these goals, a set of hardware and software design questions were

developed. The system then evolved according to a five-phase process, based on simulation and analysis, which addressed and resolved the hardware and software design considerations.

RRDS design began with development of a generic software-oriented, multi-backend database system architecture. This architecture was then refined into four general organizations to represent characteristics of previously developed relational multi-backend systems. The objective was to explore the merits of different approaches such as functional division, SIMD processing, MIMD processing, and the replicated computer concept. A comparison analysis was performed between the four architectures. The experiments indicated that multi-computer systems which run identical software and operate on a partitioned database, in parallel, were the best approach for satisfying our design goals. The resulting RRDS hardware organization consists of a set of replicated computers (RCs) communicating via a broadcast bus with a controller. Each RC manages an equal portion of the database stored on conventional secondary storage (disks). Each query is processed, in parallel, by all of the RCs.

Following development of the preliminary RRDS hardware organization, the software design questions were addressed. In order to successfully achieve the design goals proper software structures for data access, data placement, and directory management were designed. Three data access schemes were considered--hashing, clustering, and B_+ _trees. Analysis and

experimentation were conducted based upon design goals and system performance. The major selection criterion was response time, with memory requirements secondary in importance. The experiments showed the efficient range query capability of the B+_tree approach. Additionally, this scheme was determined to require less space and processing overhead than relational clustering.

Data placement, in a partitioned database, has a direct and profound impact on performance. Two strategies, round robin and value range partitioning (VRP), were considered and analyzed in terms of processing overhead during record insertions and performance improvement during record retrievals. Though costlier in terms of processing overhead during insertions, the VRP strategy guaranteed maximum parallelism during retrievals and was determined most appropriate for large relations. The simpler round robin scheme was found sufficient for small relations. RRDS accommodates both approaches to data placement providing the database creator flexibility in distributing the database according to relation size and user query habits.

The final software design question addressed was management of the directory. Five different directory management strategies for multi-backend database systems were evaluated with respect to directory size and performance. A parallel-processed and partitioned approach, in which the directory is evenly distributed across the RC network and directory management for each query is performed in parallel by all the RCs, was found to

best satisfy the design goals. Under this strategy directory size is minimized (less directory information is replicated) and processing time is reduced due to elimination of controller bottleneck and backend specialization problems.

Following completion of the RRDS system design a predictive performance analysis was conducted to determine the extent to which original design goals were satisfied. An RRDS simulation model was developed incorporating the relational query processing algorithms, the data access and placement mechanisms, and the directory management strategy. In addition, a hardware parametric model, a workload generator, and an experimentation plan were developed. A variety of experiments, designed to evaluate RRDS performance, were conducted with workload models representing various operating environments based on query, user, and database characteristics. Results of the performance analysis provided insight into how well the design meets the original goals as well as information about applicable operating environments. The predictive performance analysis was followed by an analytical comparison of our design with three other relational multi-backend systems. Timing expressions for Select, Join, and aggregate function queries were derived and evaluated for various parameters, providing information about the strengths and weaknesses of our design and query processing algorithms. Conclusions from the performance analysis and analytical comparison also provided a basis for future research.

Conclusions

We believe that this research meets its objectives in the design and analysis of a relational replicated database system. The product of this effort is the design for a multi-backend database system, which supports the relational data model and meets the design goals. The design includes query processing algorithms for the relational operations on the replicated computer organization. Directory management is accomplished in parallel on a B+_tree structure. The B+_trees serve as a hierarchical index on the database. The database is evenly distributed across the RC network according to a data placement strategy.

The RRDS design is extensible. Since the system is based on a replicated computer approach, requirements for more data handling can be satisfied by simply adding more RCs. In the predictive performance analysis percentage ideal goal figures revealed that the detrimental effects of increases in relation cardinality and query predicate characteristics could be offset by adding more RCs to the system. In addition, it was revealed that performance degradation, due to changes in query interarrival frequency, can be negated by adding more RCs. Response time analysis, for various workload scenarios, also indicated the absence of the controller limitation problem--a major design goal. This is due to the fact that the query processing algorithms are designed to minimize controller participation, eliminating it as a system bottleneck.

RRDS was found to perform best in retrieve and update-intensive environments (one-relation queries). It was shown that, for Select and Insert queries, as the cardinality of the target relation and the number of RCs were increased proportionally, query response time remained constant, or actually improved. Conversely, the poorest performance was observed for the two-relation queries in both the predictive and comparative analyses. As the cardinality of the target relations and number of RCs, for join queries, were increased response time began to degrade. This is attributed to the fact that most two-relation operations, particularly joins, require serial inter-RC transmission of relation fragments, an undesirable fact in multi-backend architectures with partitioned databases. In addition, our join algorithm is based on a worst case nested-loops procedure operating on whole relations. In the comparative analysis, RRDS was shown to perform approximately the same as MDBM and considerably better than RDBM (a system based upon functional specialization). Though DIRECT outperformed RRDS, for join queries on configurations consisting of few processors, it was shown that the RRDS performance is comparable to that of DIRECT when more processors are added. More importantly, these results showed that adding processors to RRDS produced proportional performance gains.

The RRDS design maximizes parallel processing to achieve improved throughput. Query execution algorithms and solutions to software design questions addressing data access, data placement,

and directory management were all developed to insure that all the RCs participate equally in processing each query. Relations in the database are partitioned into equivalence classes according to the data placement strategy. The records, of each relation, are partitioned according to descriptors specified by the database creator. By evenly distributing each partition across the RCS we ensure that every user request requires the same amount of data from all the RCs.

Use of a broadcast bus reduces the time required to accomplish system communications. The only serial actions accomplished by the RCs, during query processing, are the inter-RC transmissions of relation fragments for some of the two-relation operations (Join, Difference, and Intersection) and results transmission to the controller, via the bus. During the performance analysis the bus was never found to be a bottleneck.

RRDS relies only on off-the-shelf hardware for the controller, RCs, bus, and disk drives. All performance analysis and simulation results are based on hardware parameter models for commercially-available components. The absence of special-purpose hardware makes RRDS feasible and facilitates the addition of duplicate hardware for capacity growth and performance improvements. It also makes replication of software, on additional hardware, an easy task, enhancing system extensibility.

Suggestions For Future Research

Having completed the design of RRDS and established its feasibility and viability, prototyping effort can now begin. This effort should produce the database management software for RRDS. A first implementation could be accomplished on a single computer using a programming language's process communication facilities for simulating interprocessor communication. Following development and test of the single-computer RRDS prototype the software can be replicated on multiple computers producing a true multi-backend prototype system. Development of the RRDS prototype will serve several purposes. It will facilitate test of our design under real operating conditions and reveal new insight into the strengths and weaknesses of the design. System performance for different operating environments can be explored by running the prototype system against actual databases. In addition the prototype can be used to verify our RRDS simulation model. This is important because a verified RRDS simulation model would be useful in future development efforts, such as the addition of new capabilities to the system.

The RRDS design presented in this dissertation provides the basis for development of an actual operational system. There is much opportunity for enhancing the design in order to make the system commercially viable. The topics of database integrity and security have not been addressed in this dissertation. Both capabilities must be added to the design before an RRDS can be fielded. Integrity and security issues, such as granularity and

algorithms which exploit the parallel architecture of RRDS, should be investigated.

The query processing algorithms provided here are for the basic relational operations. More complex queries (such as nested Selects) should be incorporated into RRDS. Other issues which need to be addressed include development of algorithms to improve the performance of the system for two-relation queries, particularly Join queries, and providing the ability to perform more complicated queries such as two-relation operations with predicates. Development of these capabilities could proceed according to the approach we used to produce the original system design. The existing simulation models, once verified, can be enhanced to reflect new query processing techniques.

RRDS gives the database creator much flexibility in partitioning the database across the RCs. To a great extent the performance of RRDS will depend on how well this task is accomplished. The partitioning of relations, under the VRP strategy, according to descriptors will be primarily based on the database creator's knowledge of user query habits and the particular application environment. Simulation models should be developed to allow the database creator the flexibility of experimenting with various descriptor sets prior to actually partitioning a relation. Mechanisms can also be incorporated into RRDS that will collect information on database access. This information can then be used in reorganizing (better partitioning) the database.

APPENDIX A

RRDS DML SYNTAX

RRDS DML SYNTAX

```

<Query>          ::= <Select-Query> | <Project-Query> |  

                   <Insert-Query> | <Delete-Query> |  

                   <Update-Query> | <Join-Query> |  

                   <Union-Query> | <Diff-Query> |  

                   <Intsect-Query> | <Agg-Query>

<Agg-Query>      ::= <Count-Agg> | <Sum-Agg> | <Min-Agg> |  

                   <Max-Agg> | <Ave-Agg>

<Select-Query>   ::= SELECT [ALL] FROM <Target-Relation>  

                   [WHERE  

                    (<Specifier>)]  

<Project-Query>  ::= PROJECT (<Attr-List>) FROM  

                   <Target-Relation>  

                   [WHERE  

                    (<Specifier>)]  

<Insert-Query>   ::= INSERT (<Record>) INTO  

                   <Target-Relation>  

<Delete-Query>   ::= DELETE [ALL] FROM <Target-Relation>  

                   [WHERE  

                    (<Specifier>)]  

<Update-Query>   ::= UPDATE (<Modifier>) IN  

                   <Target-Relation>  

                   [WHERE  

                    (<Specifier>)]  

<Union-Query>    ::= <Target-Relation> UNION  

                   <Target-Relation>  

<Diff-Query>      ::= <Target-Relation> DIFF  

                   <Target-Relation>  

<Intsect-Query>  ::= <Target-Relation> INTSECT  

                   <Target-Relation>  

<Join-Query>      ::= <Target-Relation> JOIN  

                   <Target-Relation>  

<Count-Agg>       ::= COUNT FROM <Target-Relation>  

                   [WHERE (<specifier>)]  

<Sum-Agg>        ::= SUM (<Attribute>) FROM  

                   <Target-Relation>  

                   [WHERE (<specifier>)]
```

```

<Min-Agg>      ::= MIN (<Attribute>) FROM
                  <Target-Relation>
                  [WHERE (<specifier>)]

<Max-Agg>      ::= MAX (<Attribute>) FROM
                  <Target-Relation>
                  [WHERE (<specifier>)]

<Ave-Agg>      ::= AVERAGE (<Attribute>) FROM
                  <Target-Relation>
                  [WHERE (<specifier>)]

<Target-Relation> ::= <String>

<Record>        ::= <Record-Segment>

<Record-Segment> ::= <Attr-Value-Pair> |
                     <Attr-Value-Pair>,<Record-Segment>

<Attr-Value-Pair> ::= <Attribute> <Assign-Op> <Value>

<Attr-List>      ::= <Attribute> | <Attribute>,<Attr-List>

<Attribute>      ::= <String>

<Specifier>      ::= <Conjunction> | <Conjunction> OR
                     <Specifier>

<Conjunction>   ::= (<Pred-List>)

<Pred-List>      ::= <Predicate> | <Predicate> AND
                     <Pred-List>

<Predicate>      ::= <Attribute> <Rel-Op> <Value>

<Modifier>       ::= <Assign-Statement-List>

<Assign-Statement-List> ::= <Assign-Statement> |
                           <Assign-Statement>,<Assign-Statement-List>

<Assign-Statement> ::= <Attribute> <Assign-Op> <Arithm-Expression>

<Arithm-Expression> ::= <Term> |
                        <Arithm-Expression> <Add-Op> <Term>

<Term>           ::= <Factor> | <Term> <Mul-Op> <Factor>

<Factor>         ::= <Attribute> | <Value> |
                        (<Arithm-Expression>)

<Value>          ::= <String> | <Number>

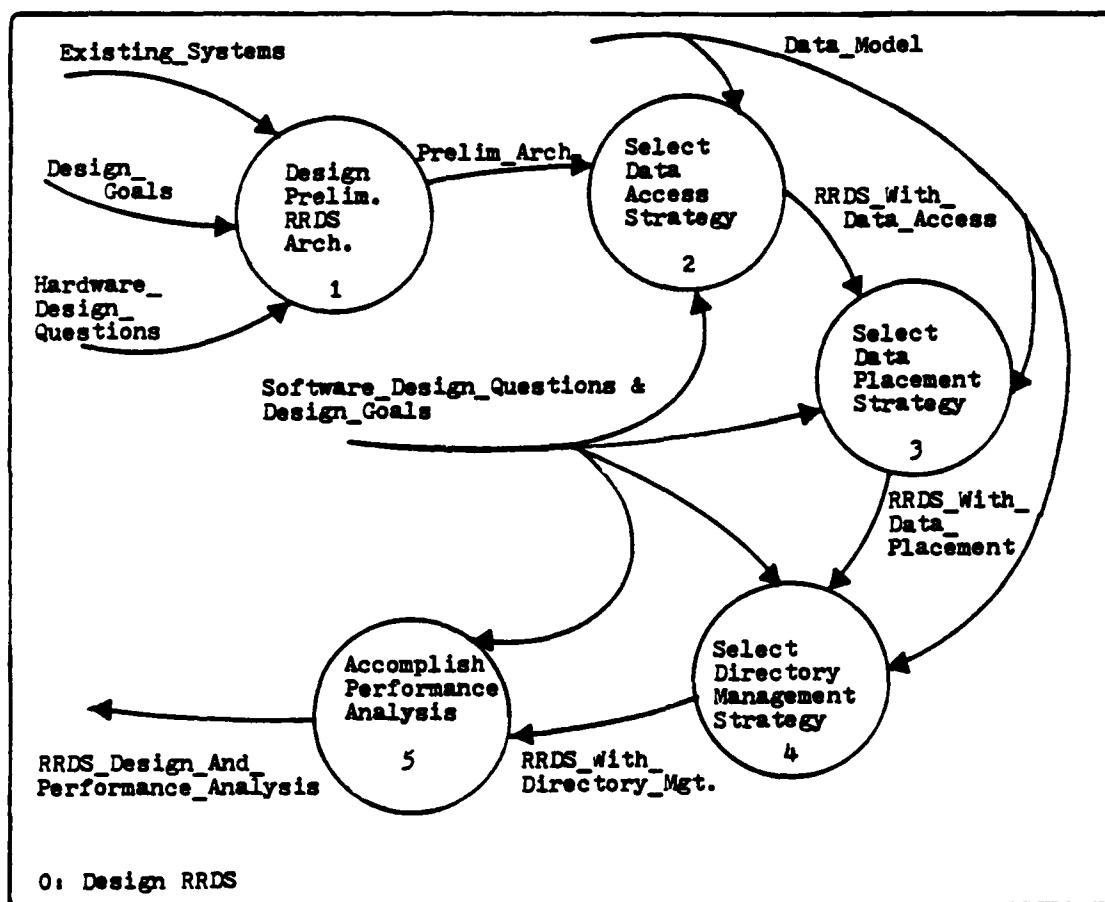
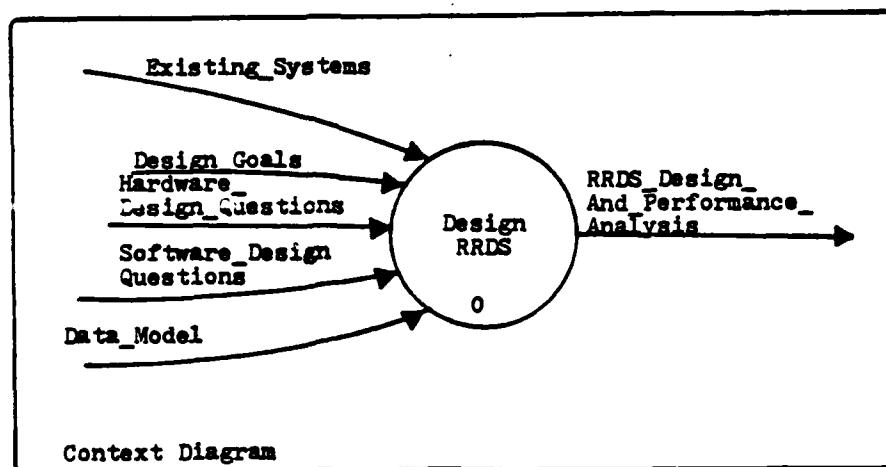
<String>          ::= <Uc-Letter> | <Uc-Letter> <Alph-Num>

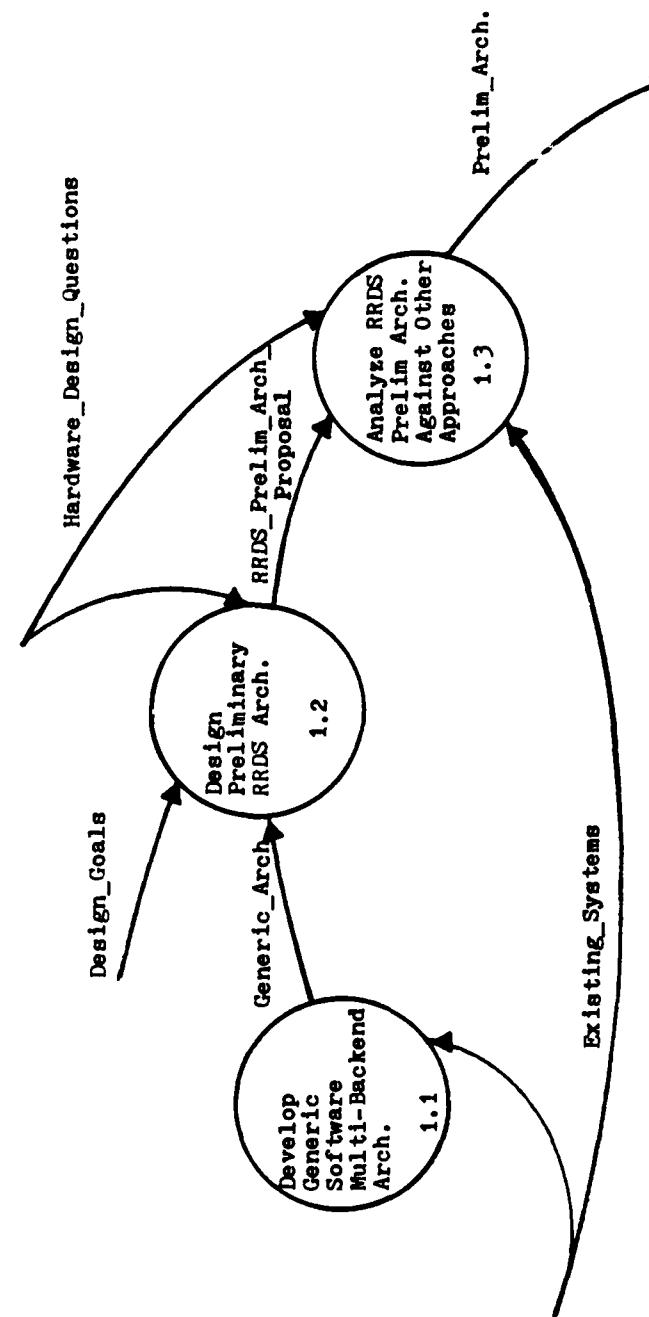
```

```
<Alph-Num>      ::= <Letter> | <Digit> |  
                  <Letter> <Alph-Num> |  
                  <Digit> <Alph-Num>  
  
<Number>        ::= <Integer> | <Real> |  
                  <Add-Op> <Integer> | <Add-Op> <Real>  
  
<Real>          ::= <Integer>.<Integer>  
  
<Integer>        ::= <Digit> | <Digit> <Integer>  
  
<Letter>         ::= <Uc-Letter> | <Lc-Letter>  
  
<Uc-Letter>     ::= A | B | C | ... | Z  
  
<Lc-Letter>     ::= a | b | c | ... | z  
  
<Digit>          ::= 0 | 1 | 2 | ... | 9  
  
<Add-Op>         ::= + | -  
  
<Mul-Op>         ::= * | /  
  
<Rel-Op>         ::= < | <= | > | >= | <> | =  
  
<Assign-Op>      ::= :=
```

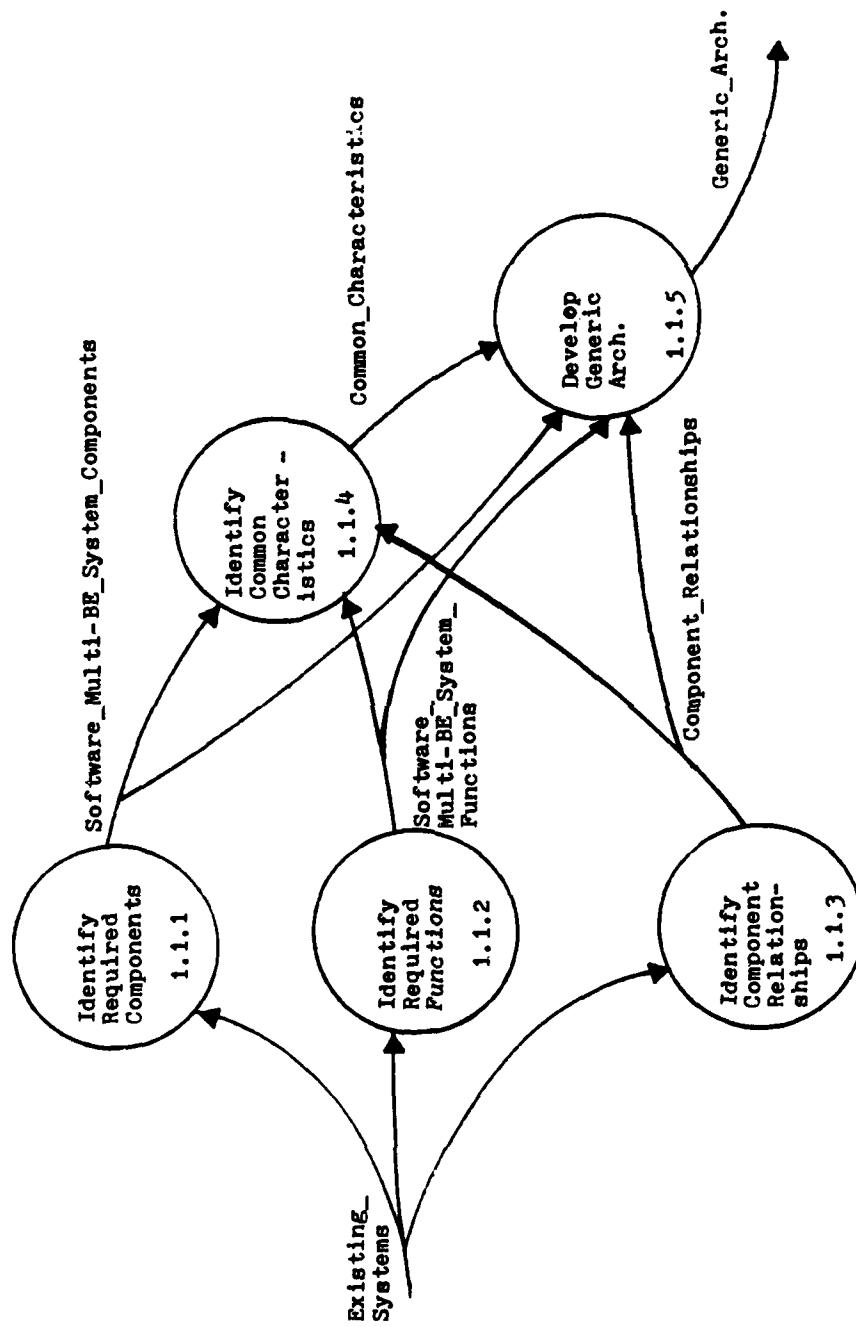
APPENDIX B

DATA FLOW DIAGRAM FOR THE RRDS DESIGN METHODOLOGY MODEL

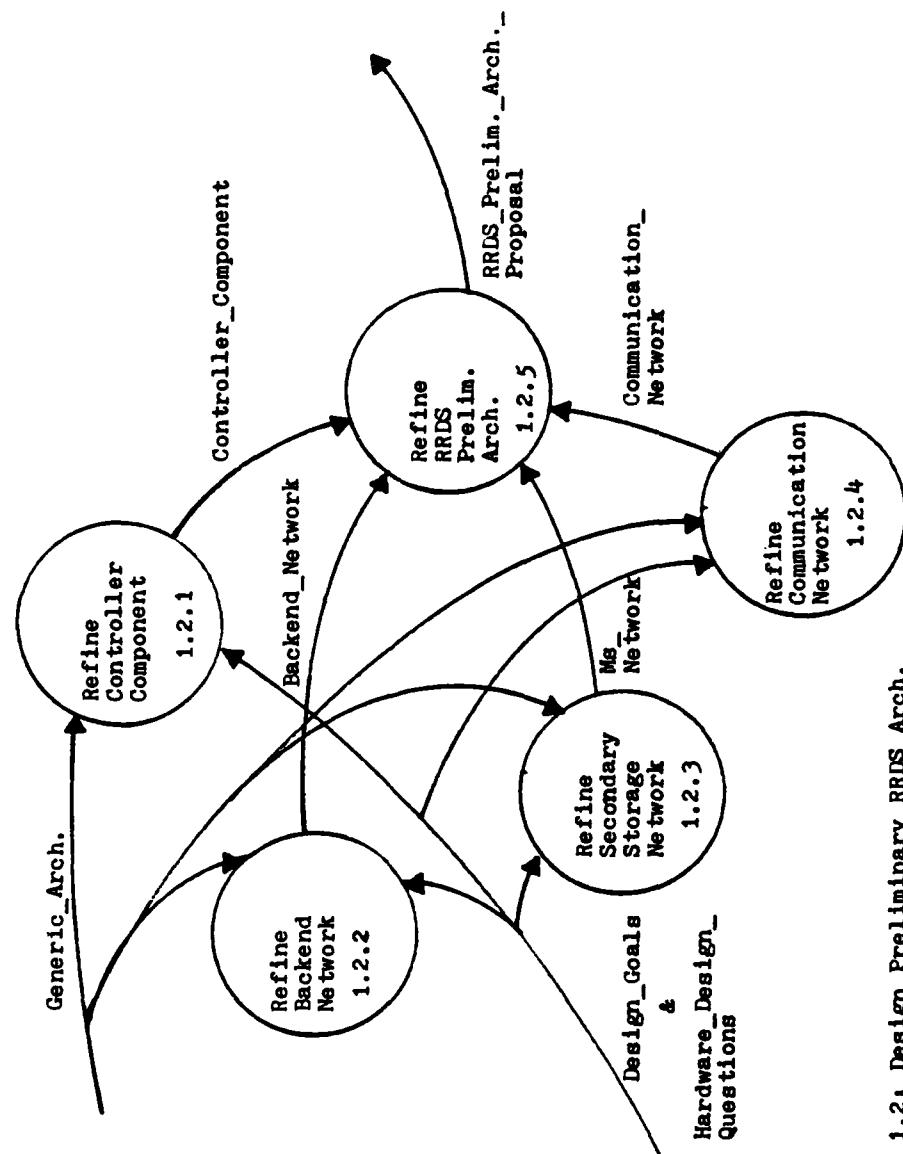




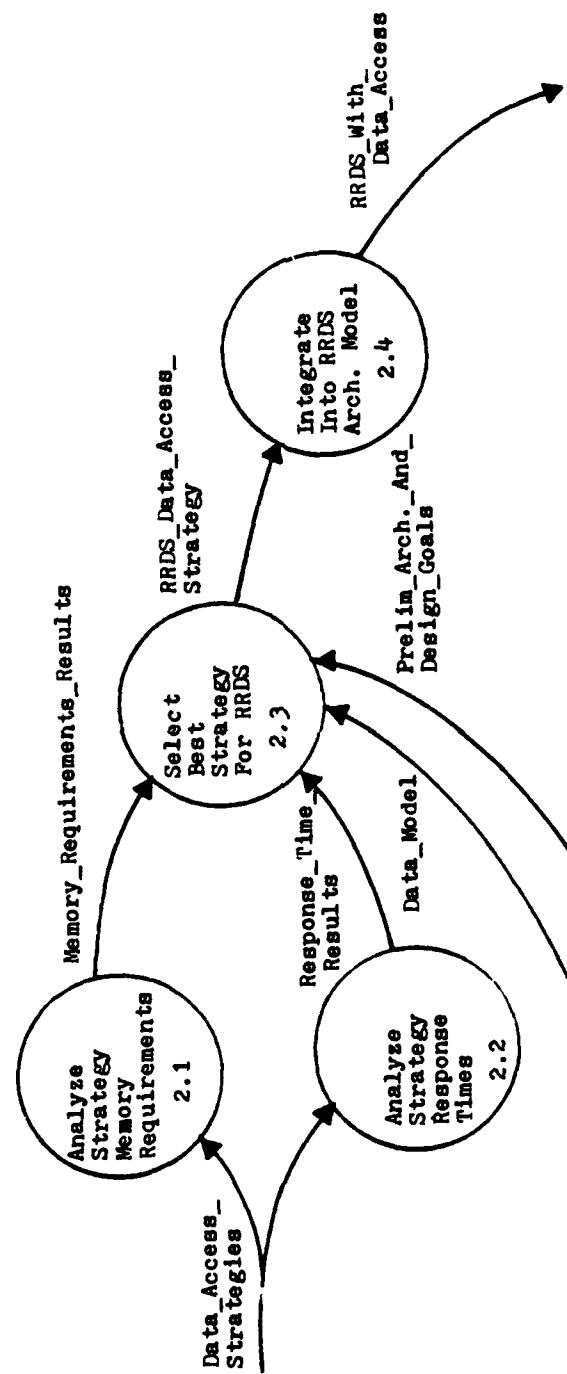
1. Design Preliminary RRDs Architecture



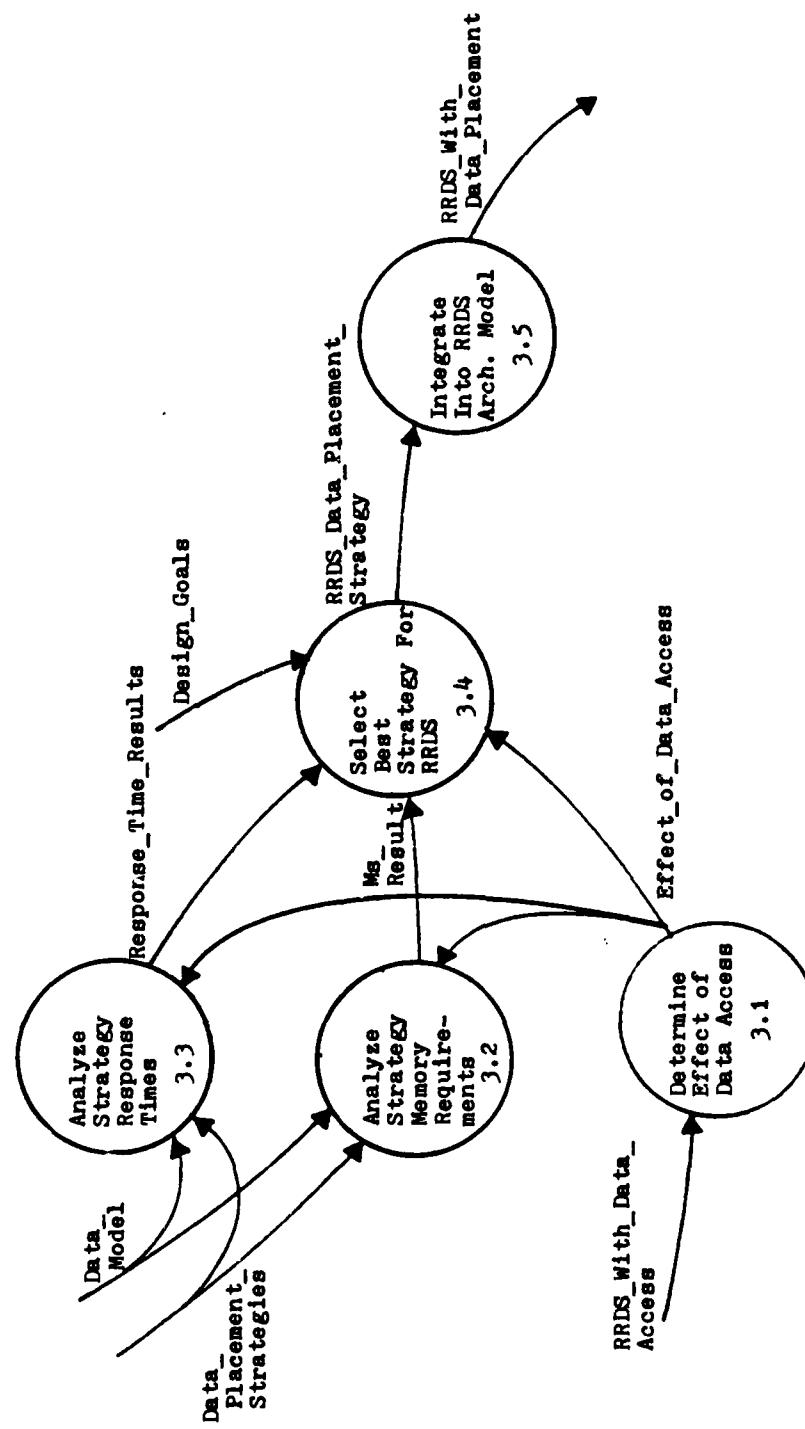
1.1: Develop Generic Software Multi-BE Architecture



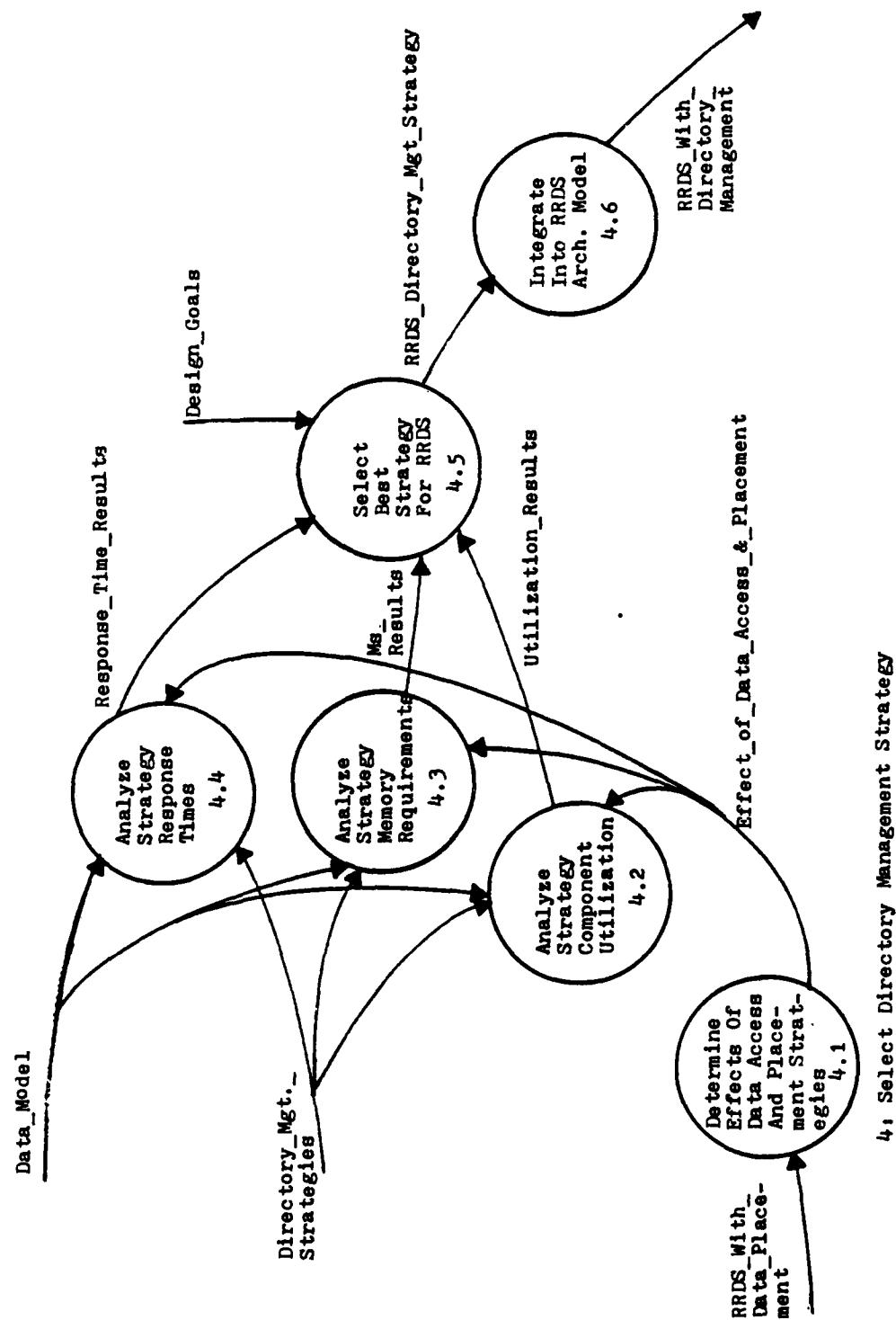
1.2: Design Preliminary RRDS Arch.



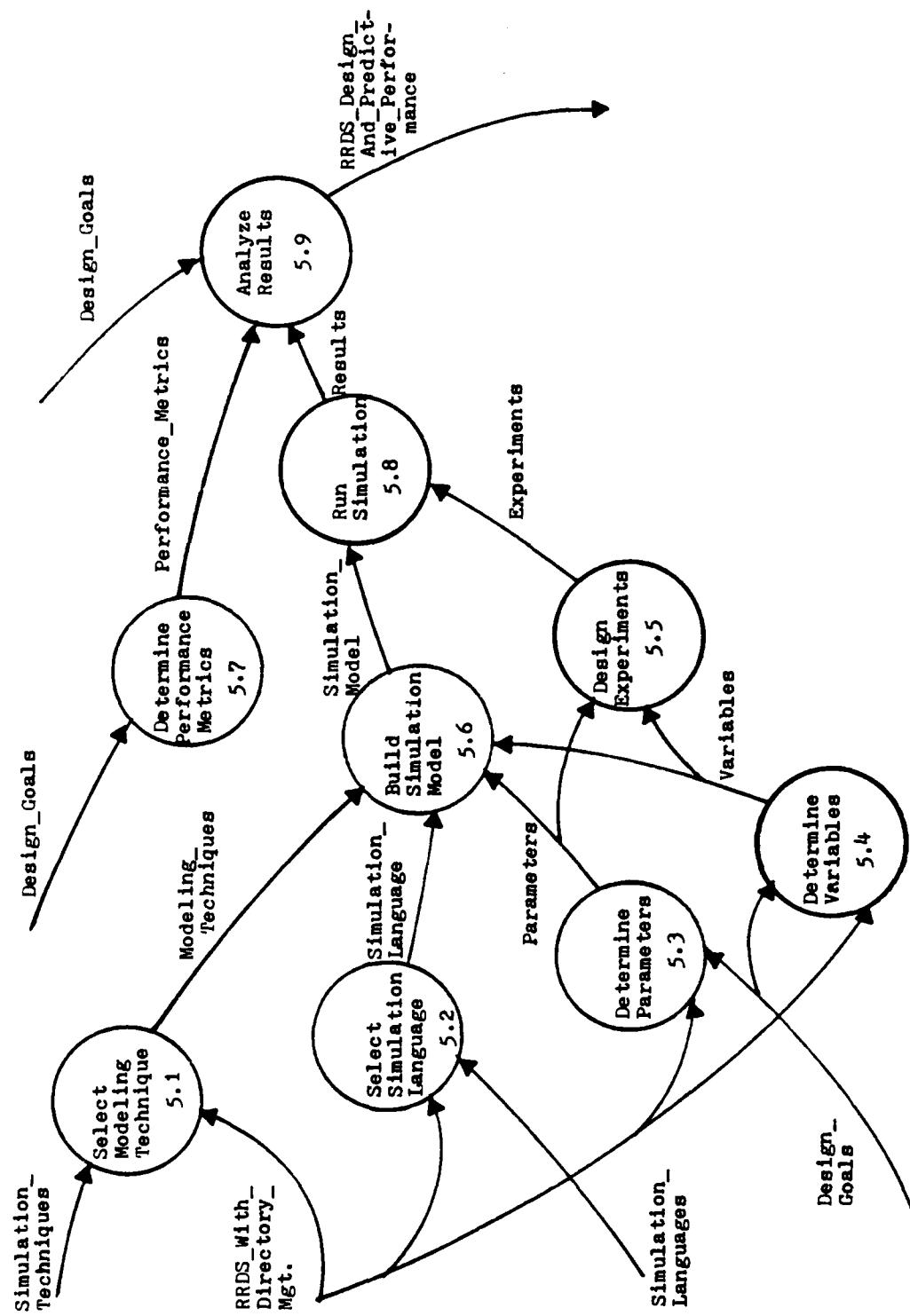
2.1 Select Data Access Strategy



3: Select Data Placement Strategy

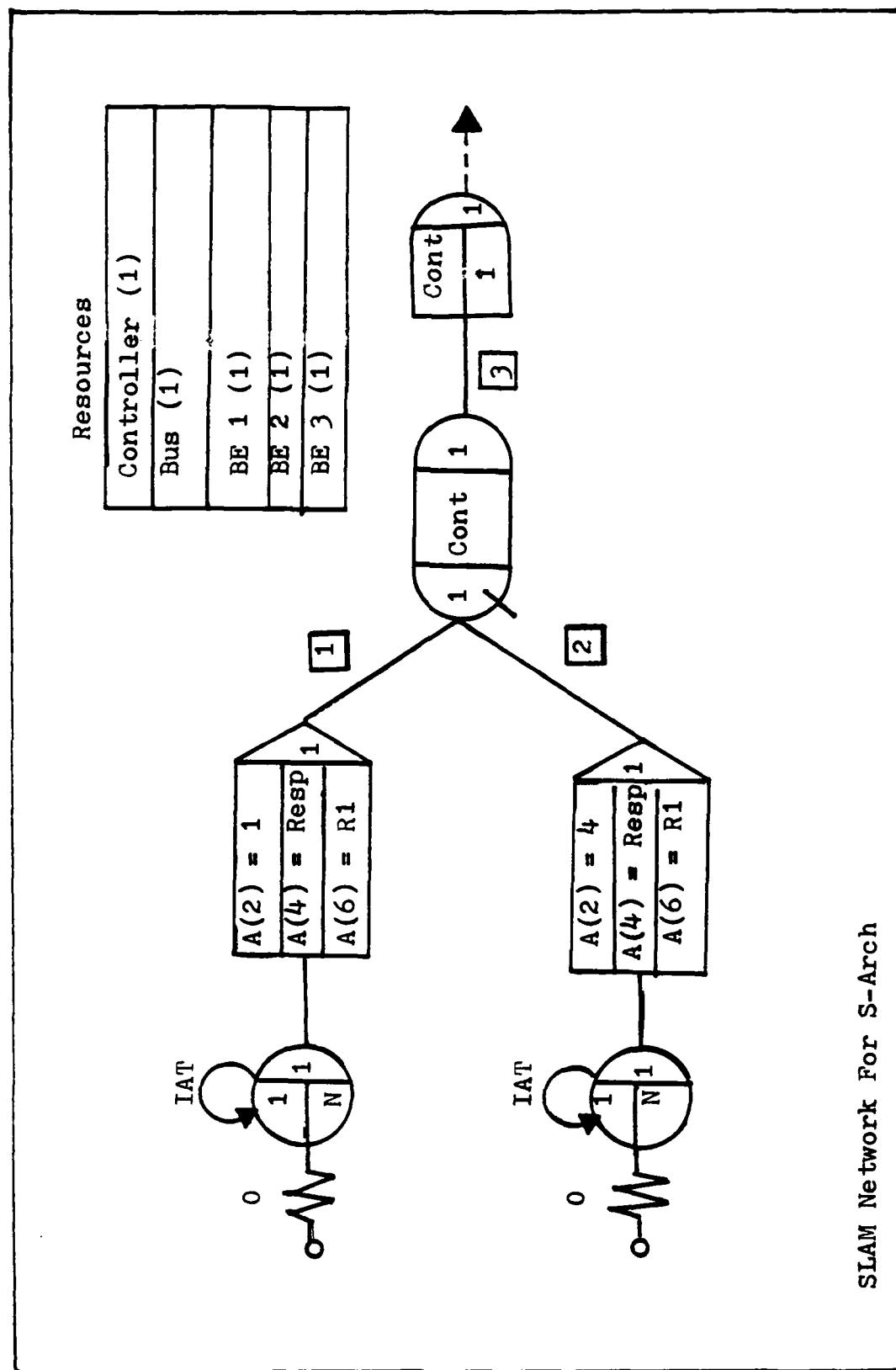


4: Select Directory Management Strategy

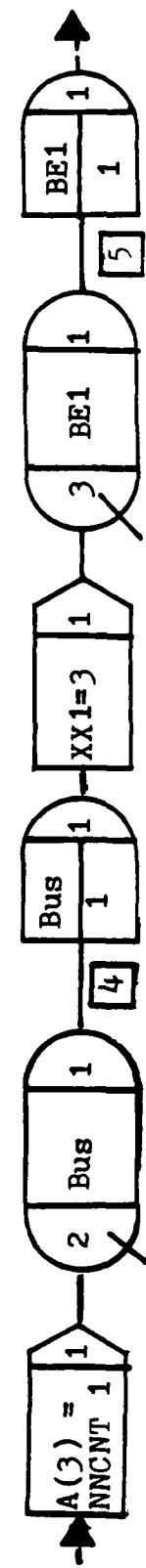


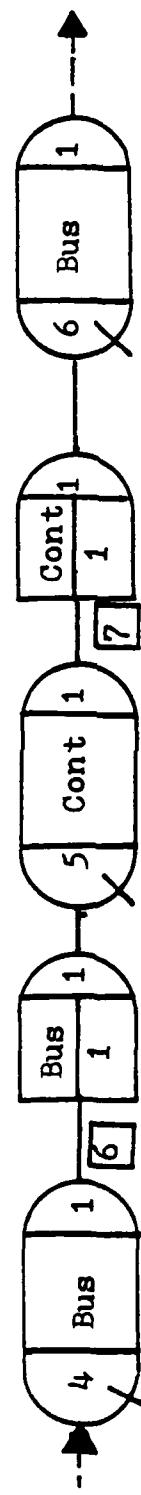
APPENDIX C

SLAM NETWORKS FOR ARCHITECTURAL COMPARISON ANALYSIS



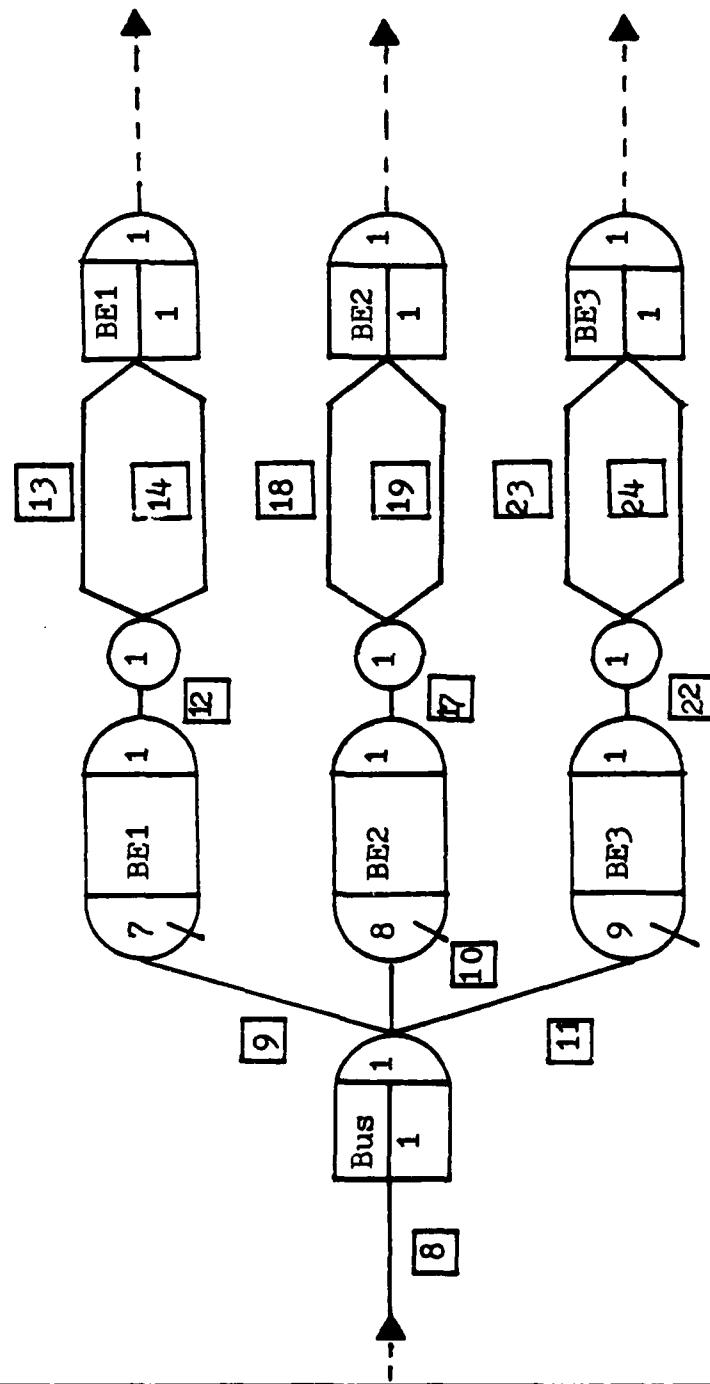
SLAM Network For S-Arch

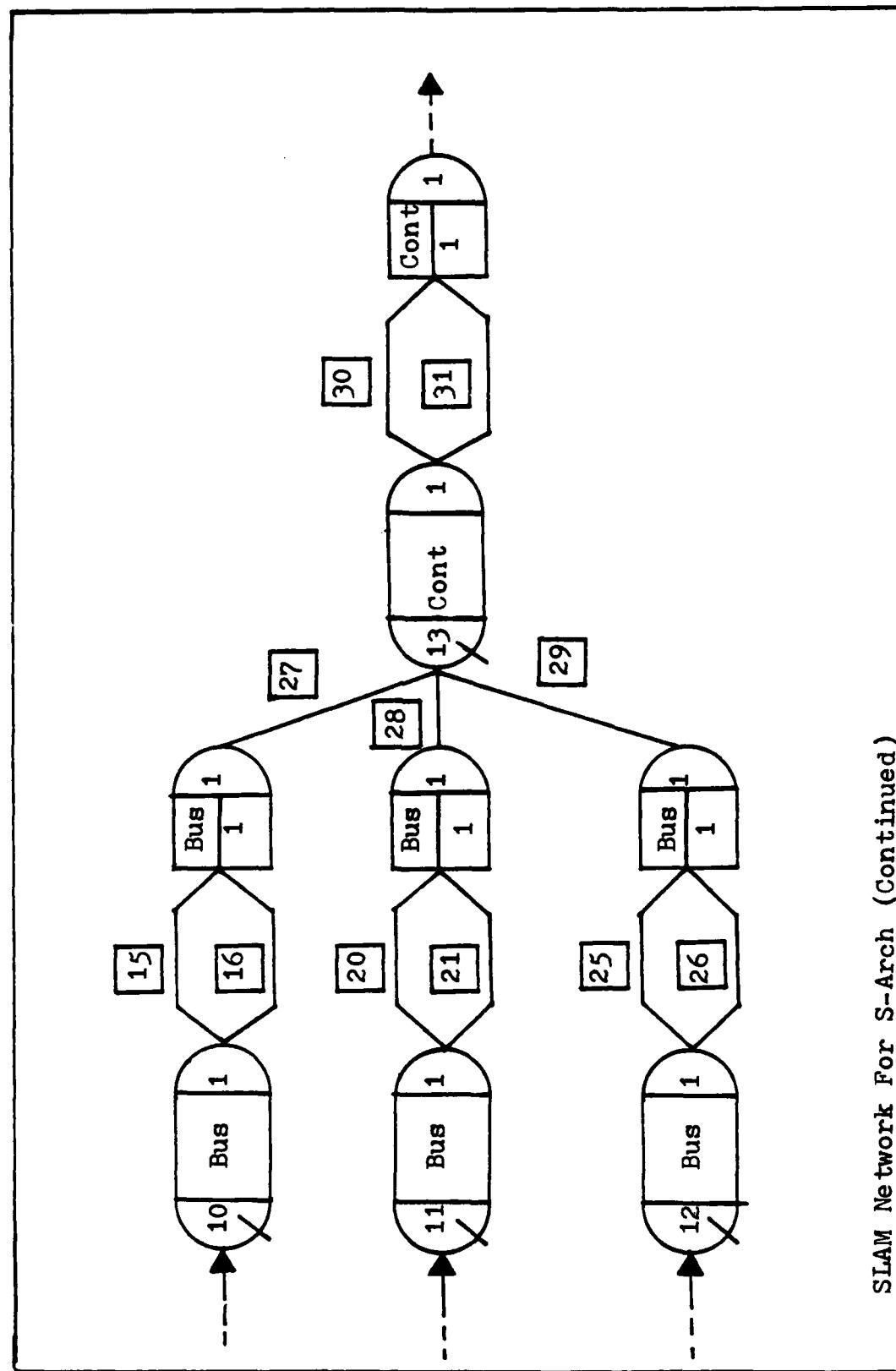


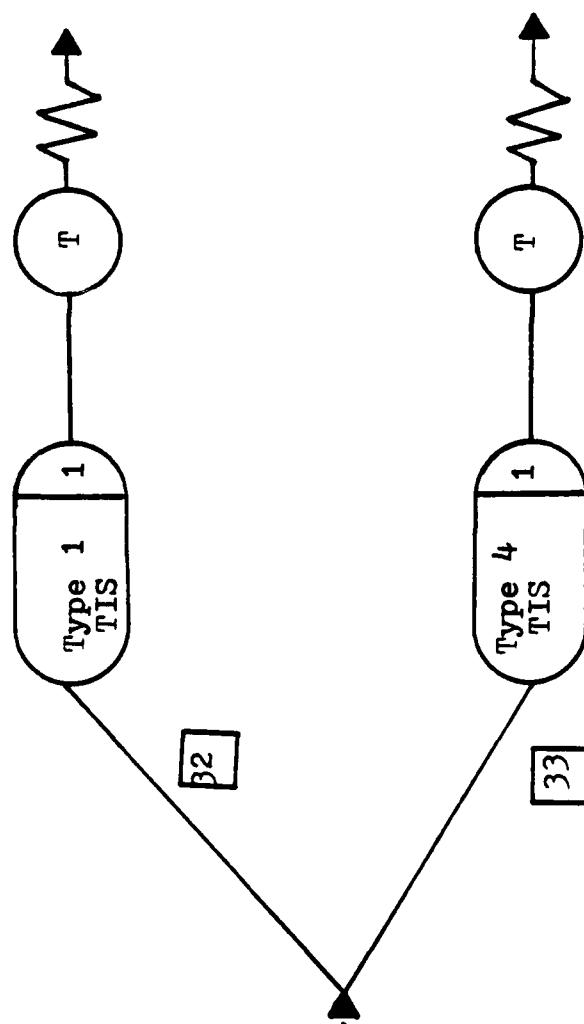


SLAM Network For S-Arch (Continued)

SLAM Network For S-Arch (Continued)







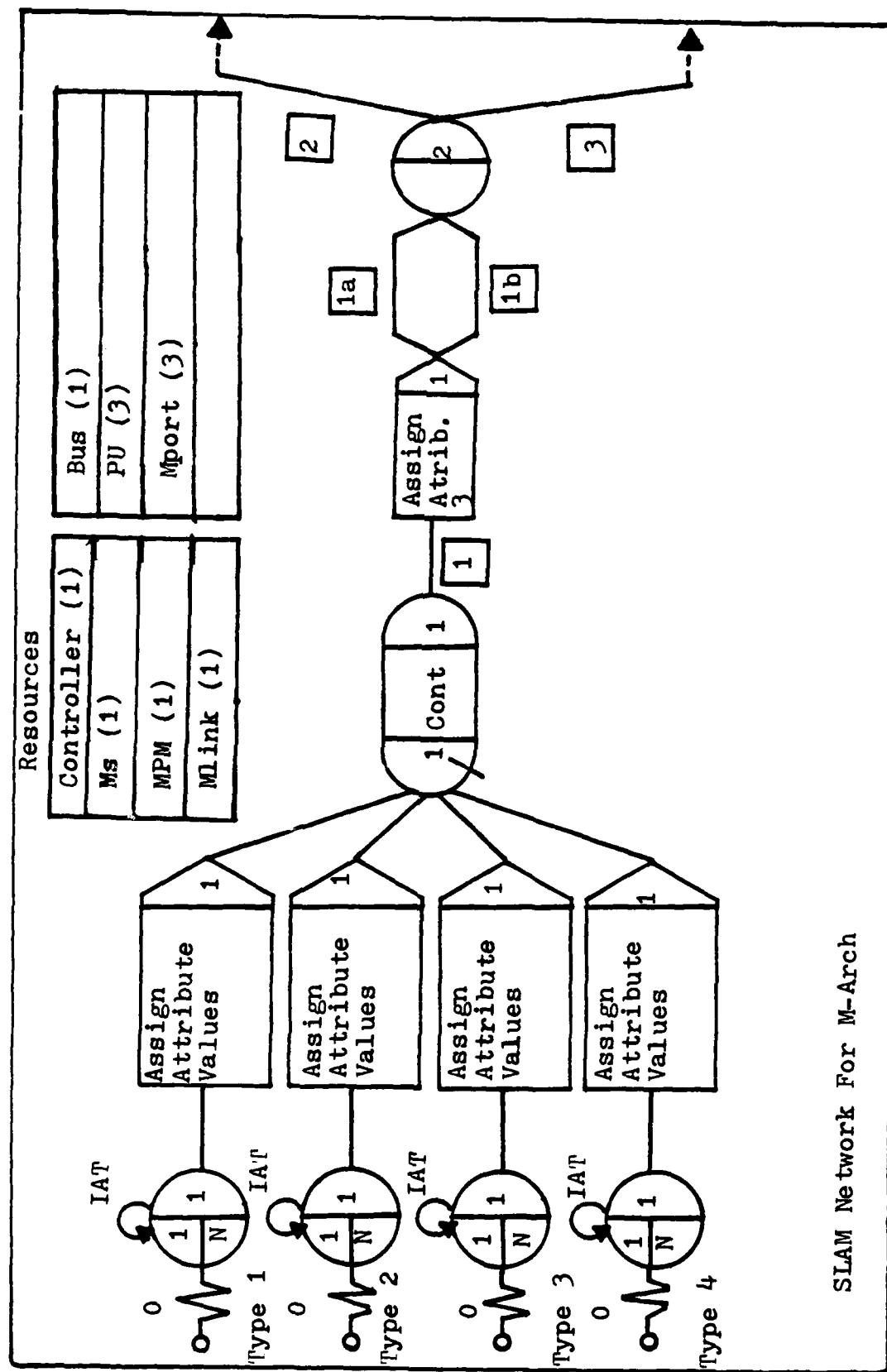
SLAM Network For S-Arch (Continued)

S-Arch Simulation Model Attributes

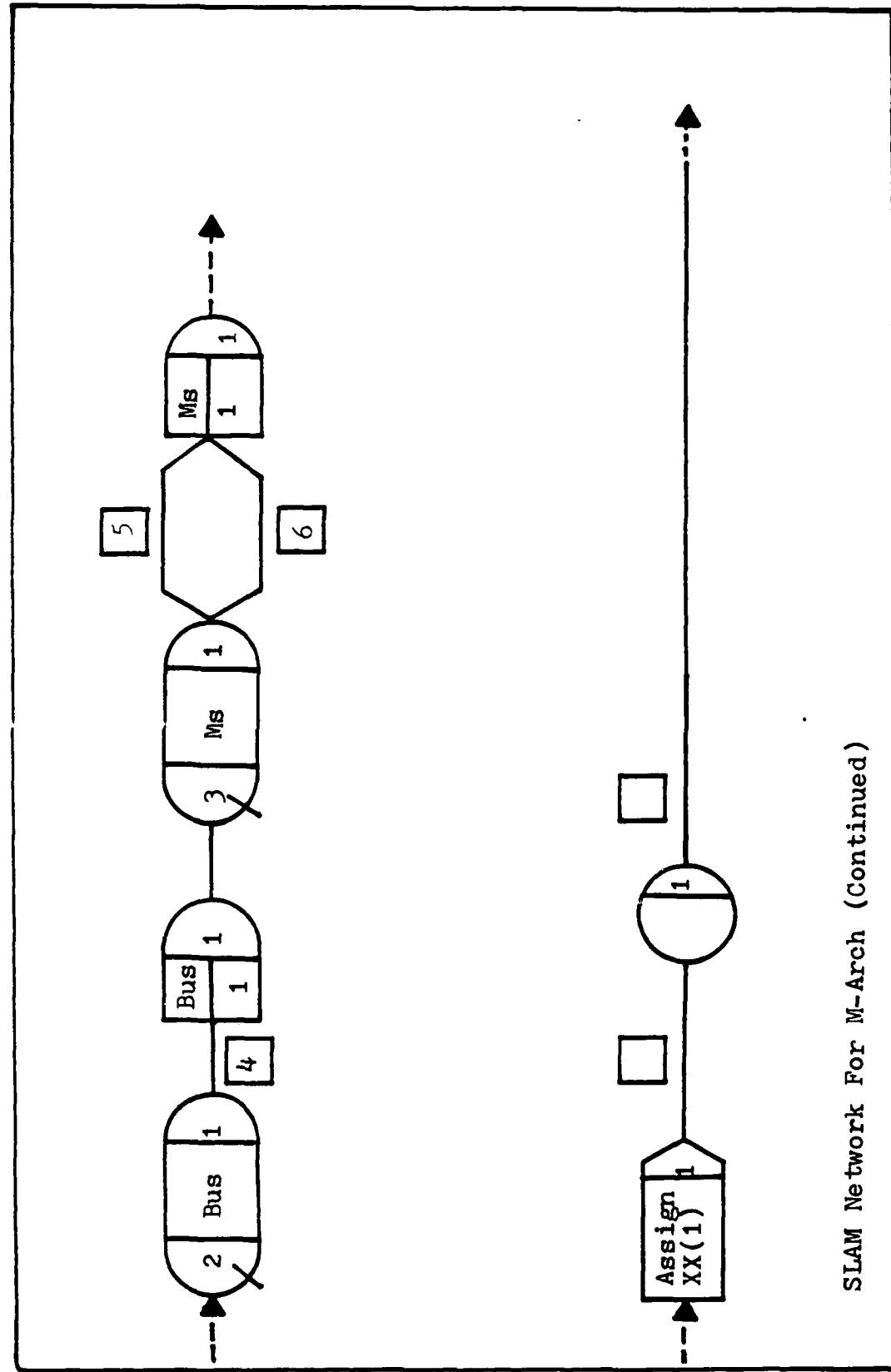
<u>Attribute</u>	<u>Description</u>
1	Mark Time
2	Query Type
3	Query Identifier
4	Resp1
6	R1
xx(1)	n

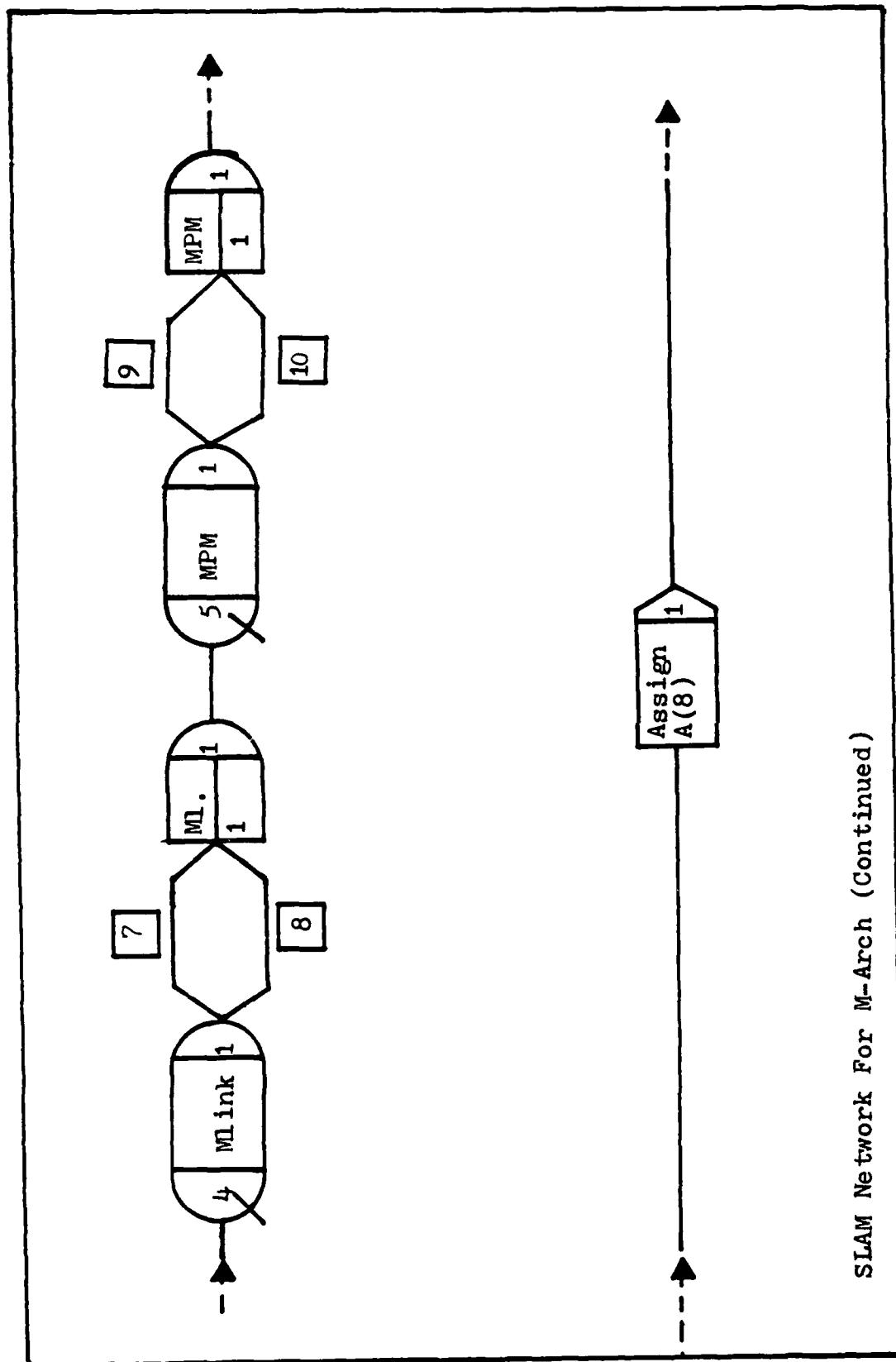
Activities In S-Arch SLAM Simulation Model

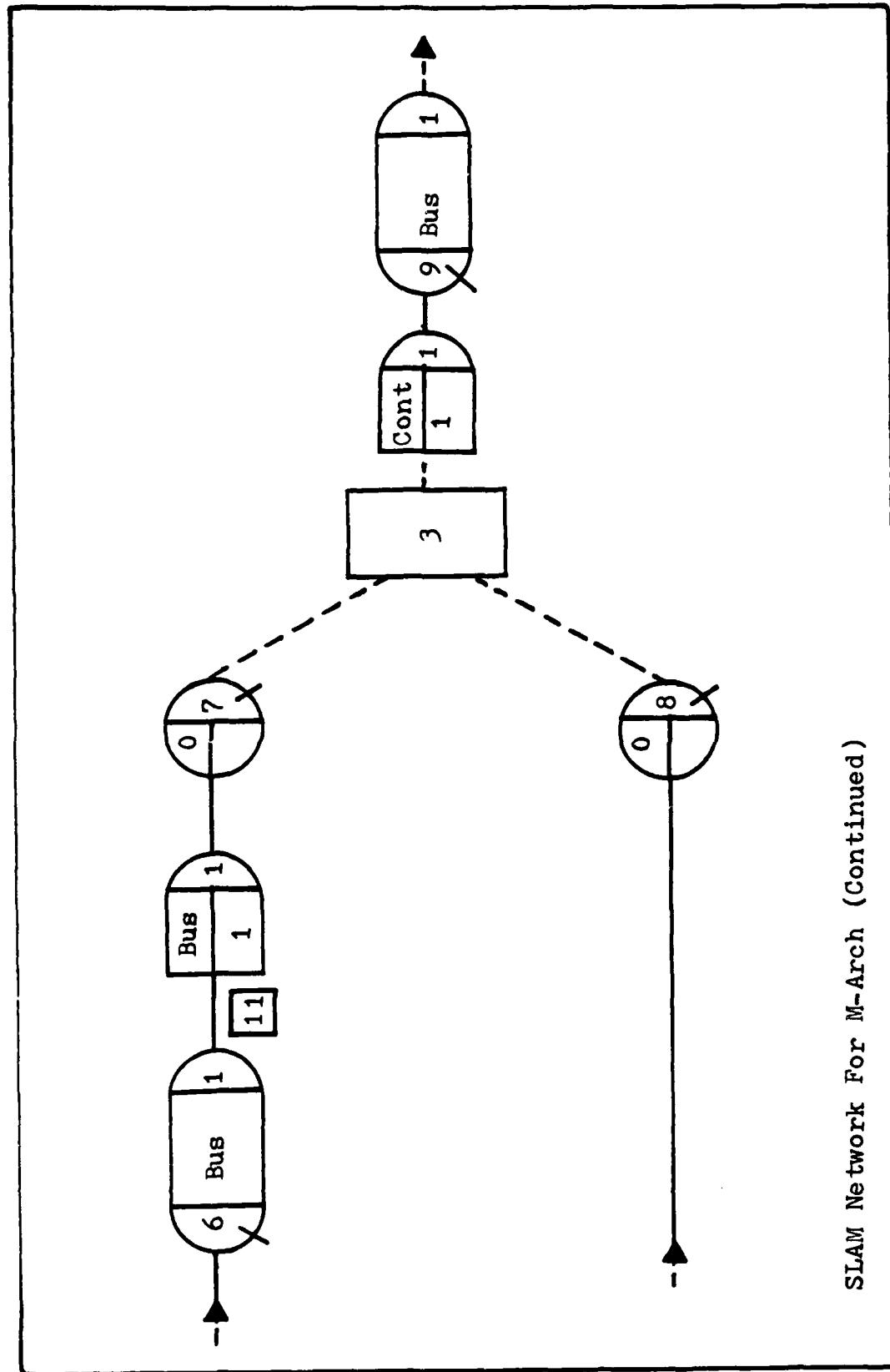
<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	0
2	0
3	Tparse
4	Tmtrans
5	xx(1)*Ta
6	Tmtrans
7	xx(1)*Tp
8	Tmtrans
9	variable probability
10	variable probability
11	variable probability
12,17,22	A(6)*Ta + A(4)*Tr
13,18,23	A(4)*Tp when A(2) = 1
14,19,24	Tp when A(2) = 4
15,20,25	A(4)*Tmtrans when A(2) = 1
16,21,26	Tmtrans when A(2) = 4
27,28,29	0
30	A(4)*Tp when A(2) = 1
31	Tp when A(2) = 4
32	A(2) = 1
33	A(2) = 4

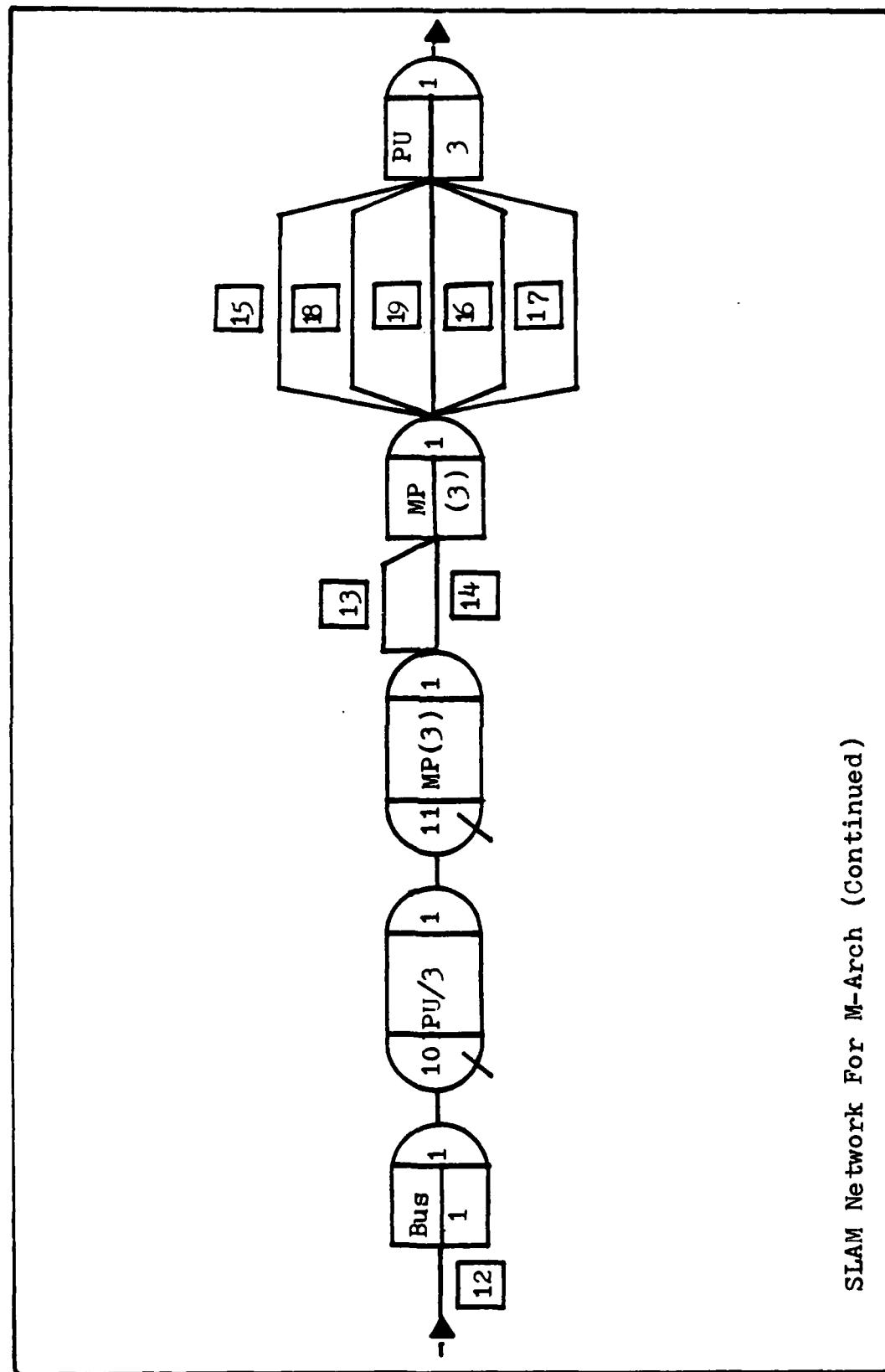


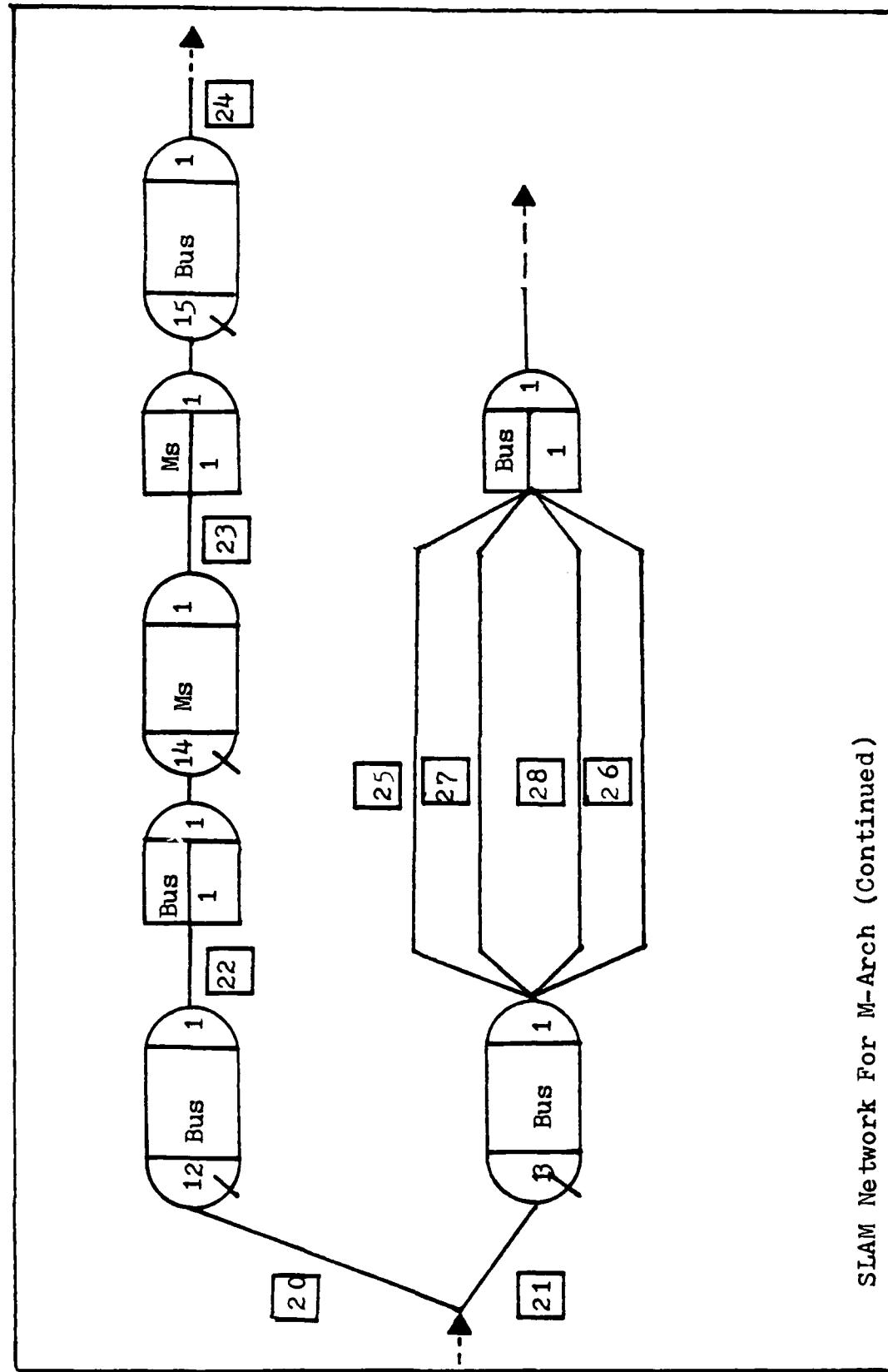
SLAM Network For M-Arch

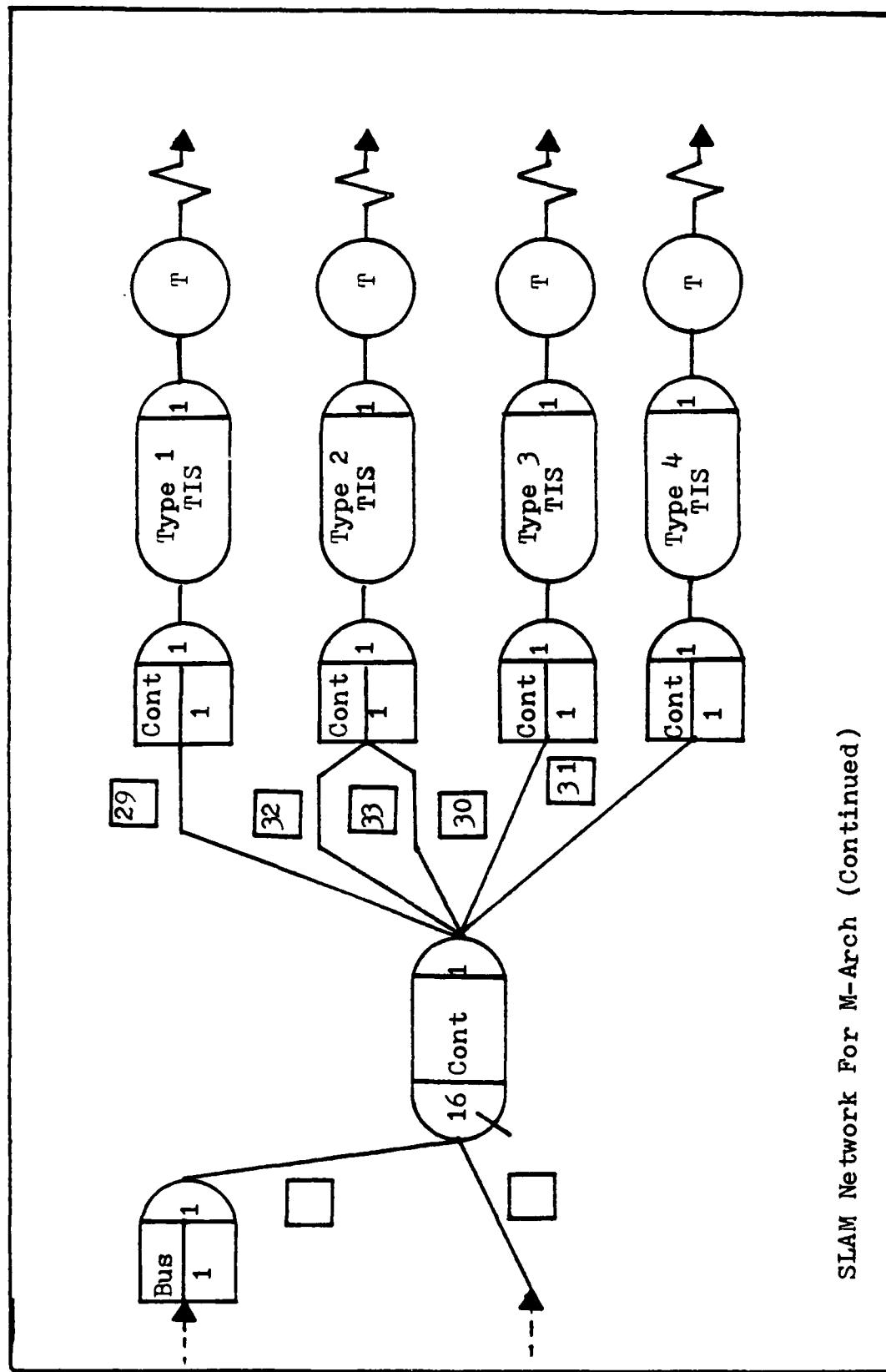












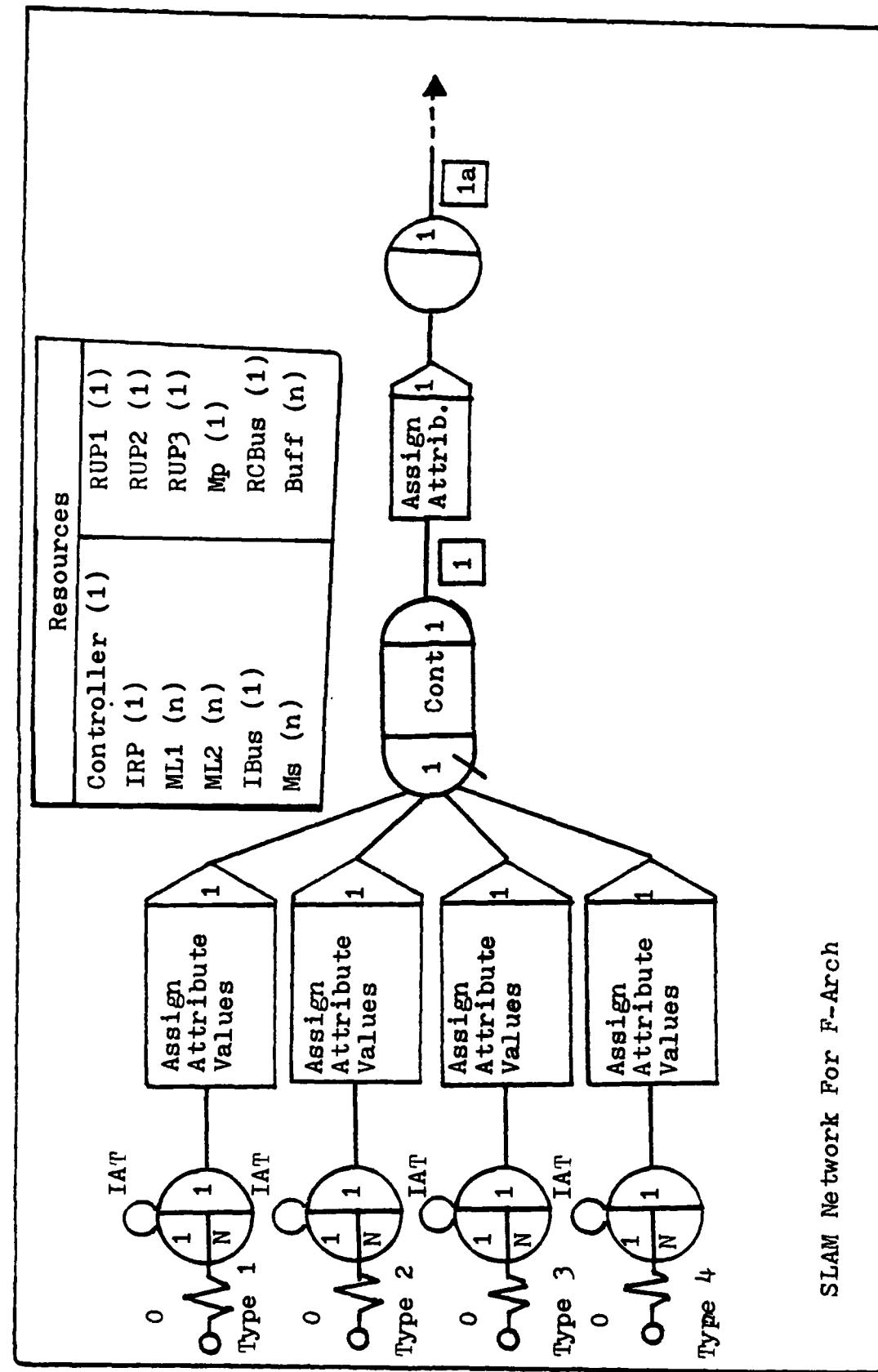
M-Arch Simulation Model Attributes

Attribute	Description
1	Mark Time
2	Query Type
3	Query Identifier
4	Resp1
5	Resp2
6	R1
7	R2
xx(1)	n
8	p

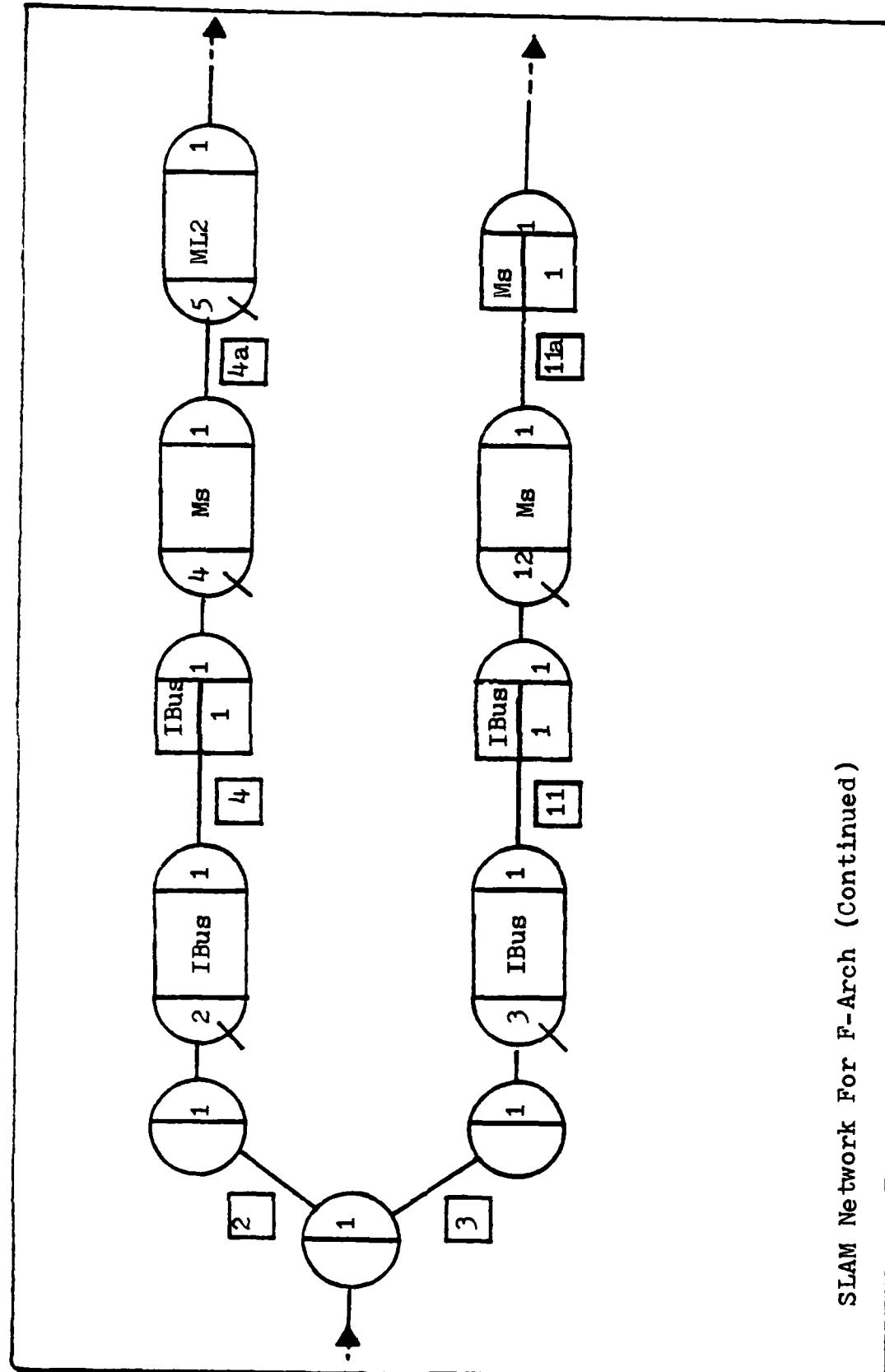
Activities In M-Arch SLAM Simulation Model

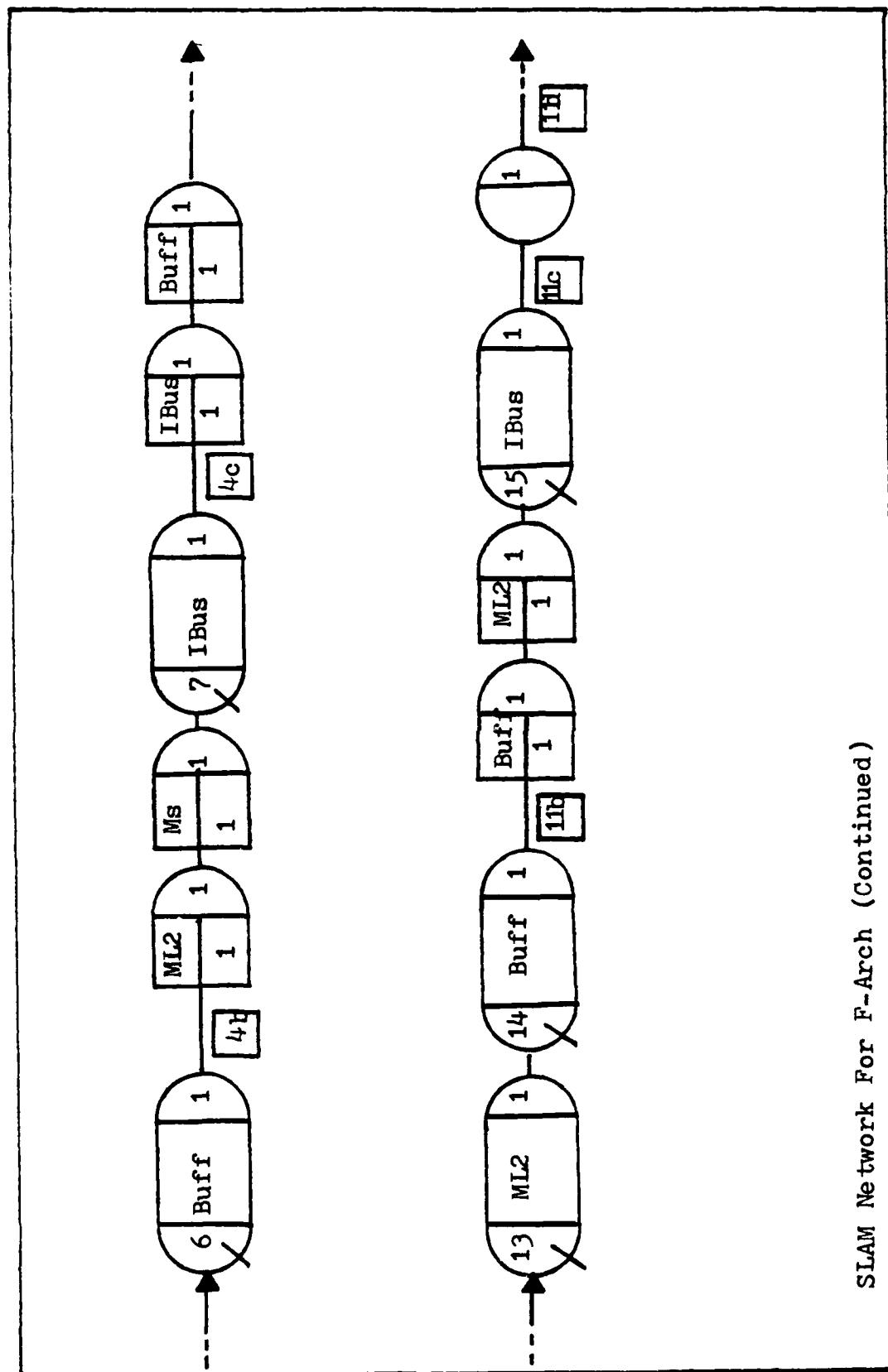
Activity	Duration Or Branching Condition
1	Tparse
1a	Tp when A(2)=1 or A(2)=4
1b	2Tp when A(2)=2 or A(2)=3
2,3	0
4	Tmtrans
5	A(6)*[Ta+Tr] when A(2)=1 or A(2)=4
6	[A(6)+A(7)]*[Ta+Tr] when A(2)=2 or A(2)=3
7	A(6)*Trtrans when A(2)=1 or A(2)=4
8	[A(6)+A(7)]*Trtrans when A(2)=2 or A(2)=3
9	A(6)*Tc when A(2)=1 or A(2)=4
10	[A(6)+A(7)]*Tc when A(2)=2 or A(2)=3
11	Tmtrans
12	Tmtrans
13	[A(6)/A(8)]*Tc when A(2)=1 or A(2)=4
14	[(A(6)+A(7))/A(8)]*Tc when A(2)=2 or A(2)=3
15	[A(6)/A(8)]*Tp+[A(4)/A(8)]*Tp when A(2)=1
16	[(A(6)+A(7))/A(8)]*Tp when A(2)=3
17	[A(6)/A(8)]*Tp when A(2)=4
18	[(A(6)/A(8))+A(7)]*Tp+[(A(4)*A(5))/A(8)]*Tp with prob .50
19	[(A(6)+A(7))/A(8)]*Tp+[A(4)/A(8)]*Tp with prob .50
20	A(2)=4

21 A(2)=1 or A(2)=2 or A(2)=3
22 A(4)*Trtrans
23 A(4)*(Ta+Tr)
24 Tmtrans
25 A(4)*Trtrans when A(2)=1
26 [A(6)+A(7)]*Trtrans when A(2)=3
27 [A(4)*A(5)]*Trtrans with prob .50
28 A(4)*Trtrans with prob .50
29 A(4)*Tp when A(2)=1
30 [A(6)+A(7)]*Tp when A(2)=3
31 A(2)=4
32 [A(4)*A(5)]*Tp with prob .50
33 A(4)*Tp with prob .50

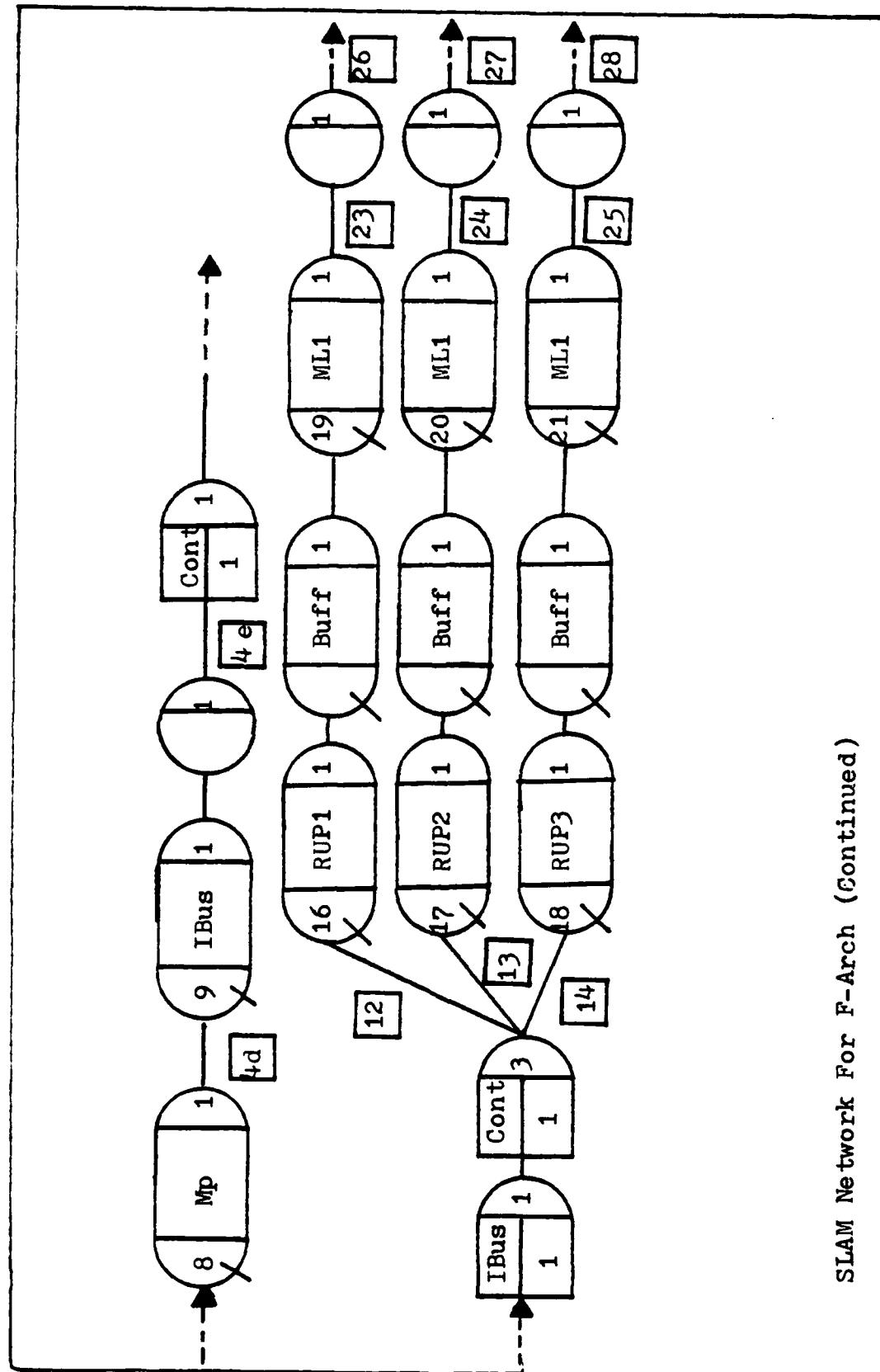


SLAM Network For F-Arch (Continued)

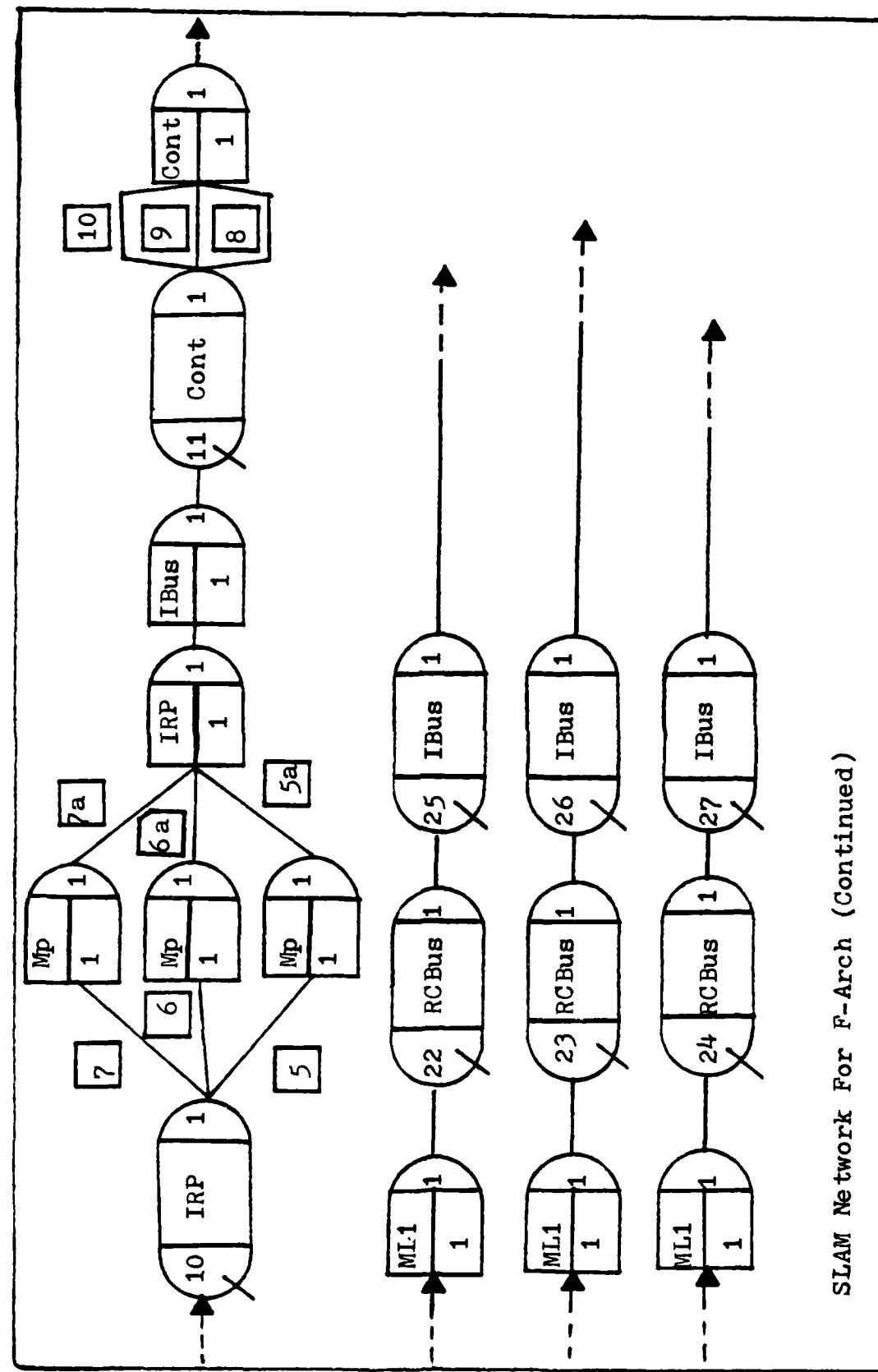




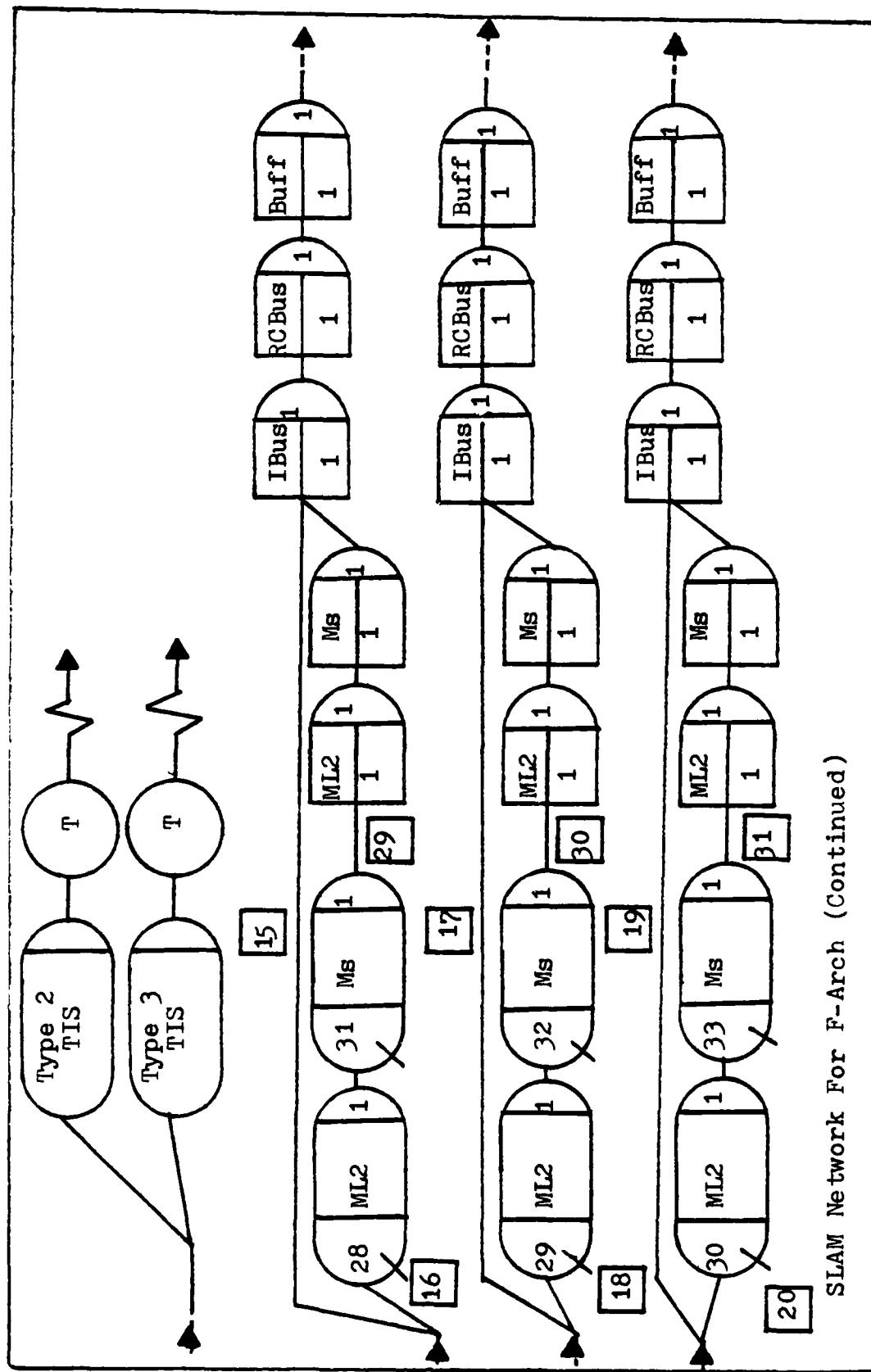
SLAM Network For F-Arch (Continued)



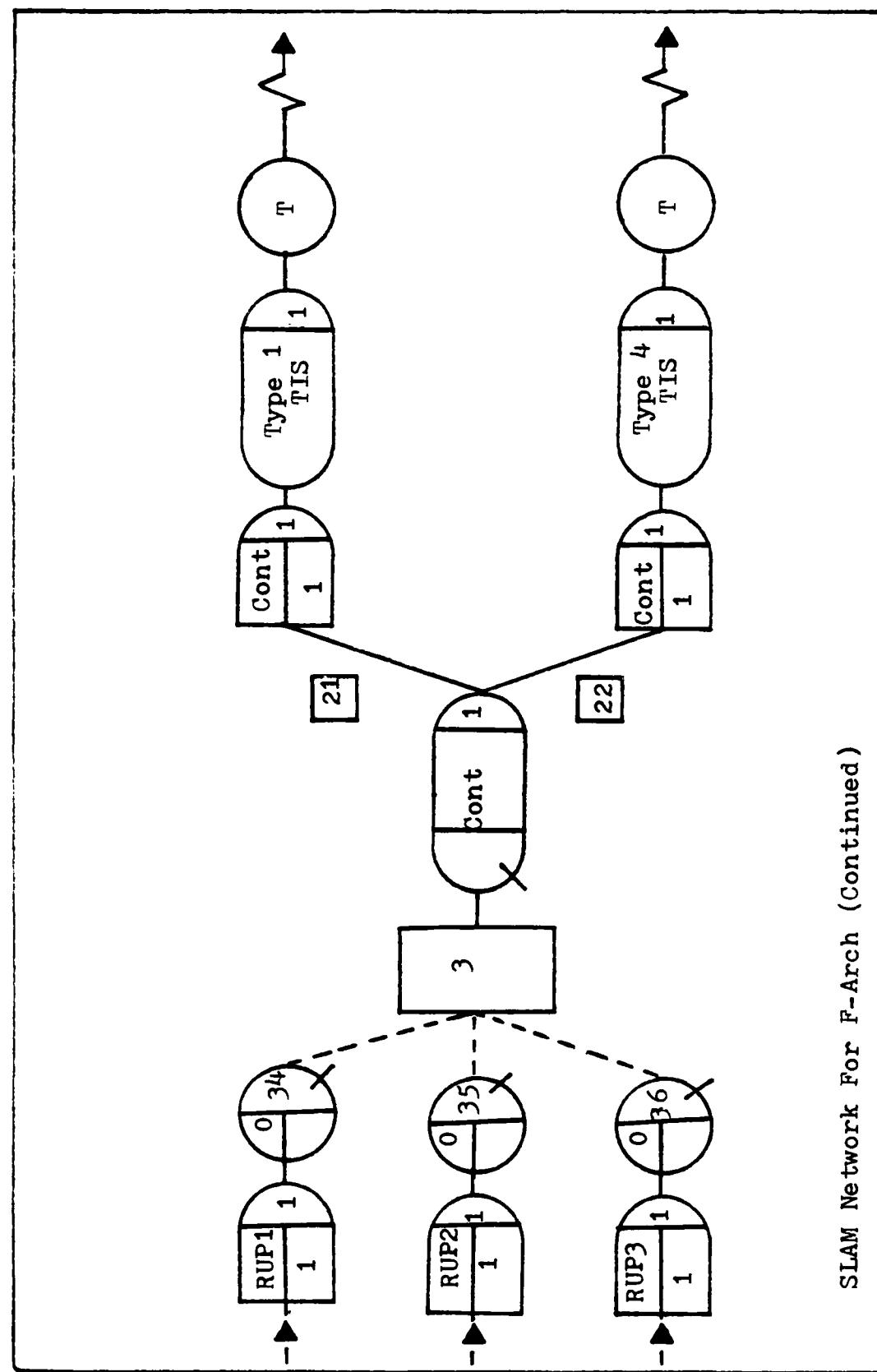
SLAM Network For F-Arch (continued)



SLAM Network For F-Arch (Continued)



SLAM Network For F-Arch (Continued)



SLAM Network For R-Arch (Continued)

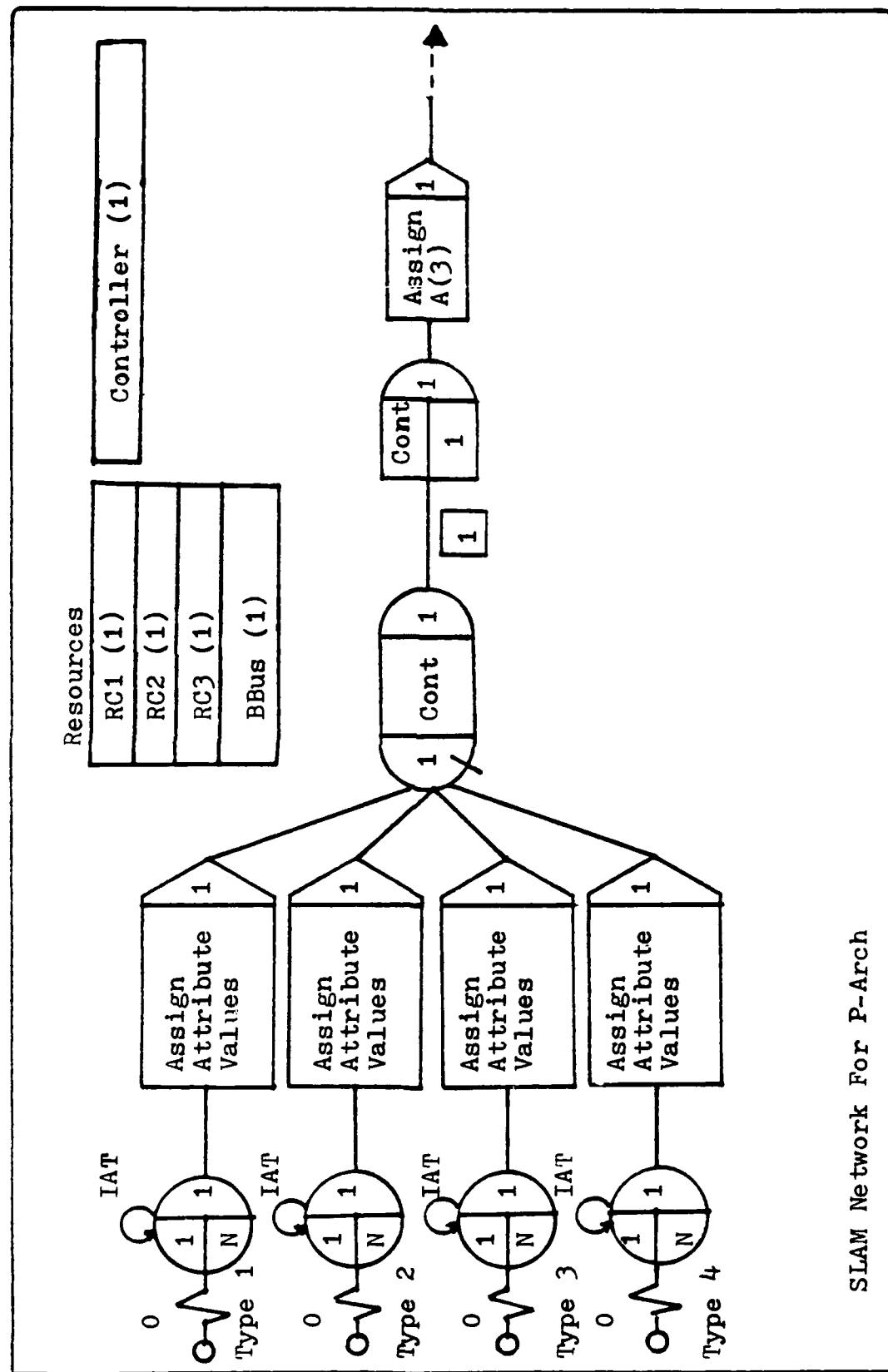
F-Arch Simulation Model Attributes

<u>Attribute</u>	<u>Description</u>
1	Mark Time
2	Query Type
3	Query Identifier
4	Resp1
5	Resp2
6	R1
7	R2
xx(1)	n

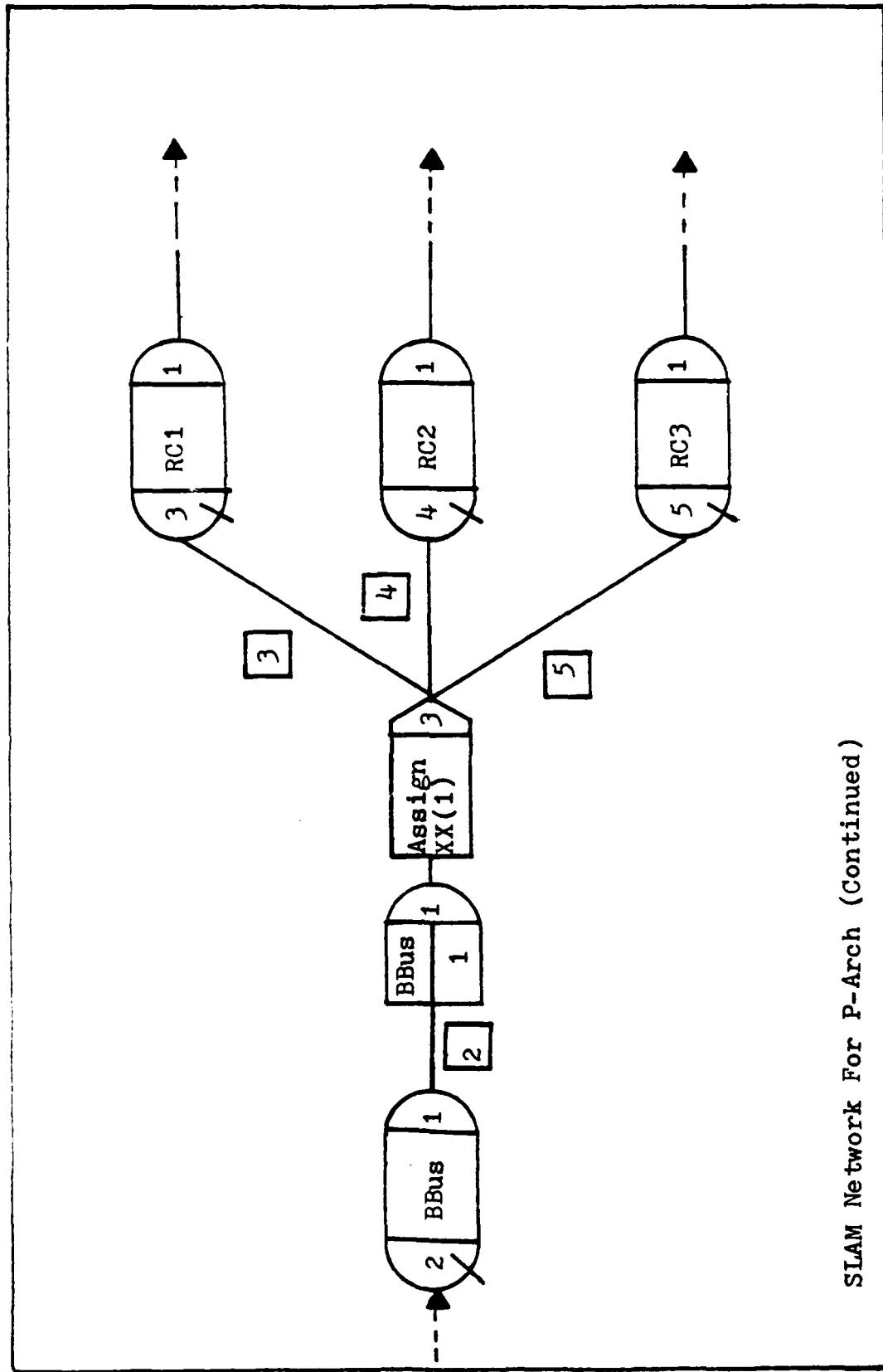
Activities In F-Arch SLAM Simulation Model

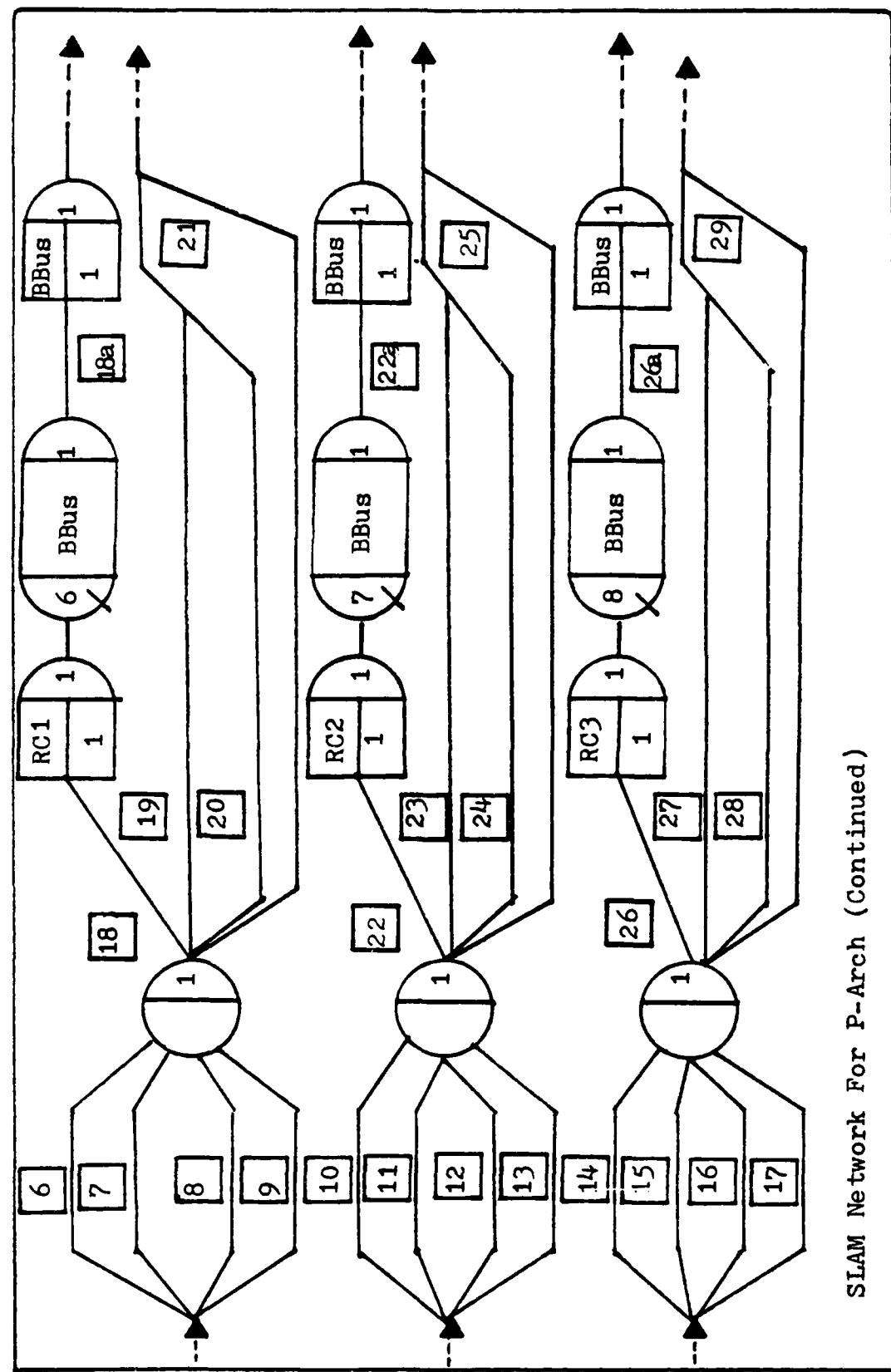
<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	Tparse
1a	[xx(1)+3]*Tp
2	3Tp when A(2)=2 or A(2)=3
3	[xx(1)+2]*Tp when A(2)=1 or A(2)=4
4	2*Tmtrans
4a	[A(6)+A(7)]*Ta + [A(6)+A(7)]*Tr
4b, 4c	[A(6)+A(7)]*Trtrans
4d	[A(6)+A(7)]*Tc
4e	2Tmtrans
5	[A(6)+A(7)]*Trtrans+[A(6)+A(7)]*Tp when A(2)=3
6	[A(6)+A(7)]*Trtrans+[A(4)*Tp] with prob .50
7	[A(6)+A(7)]*Trtrans+[A(4)*A(5)]*Tp with prob .50
5a	[A(6)+A(7)]*Trtrans
6a	A(4)*Trtrans
7a	[A(4)*A(5)]*Trtrans
8	[A(6)+A(7)]*Tp when A(2)=3
9	A(4)*Tp with prob .50
10	[A(4)*A(5)]*Tp with prob .50
11	Tmtrans
11a	A(6)*Ta+A(6)*Tr
11b	A(6)*Trtrans
11c	Tmtrans
11d	xx(1)*Tmtrans

12,13,14	0
15,17,19	$[A(4)/xx(1)]^*Trtrans$ when $A(2)=1$
16,18,20	$A(2)=4$
21	$A(4)^*Tp$ when $A(2)=1$
22	$A(2)=4$
23,24,25	$[A(6)/xx(1)]^*Trtrans$
26,27,28	$[A(4)/xx(1)]^*Tp$
29,30,31	$[A(4)/xx(1)]^*[2Trtrans+Ta+Tr]$

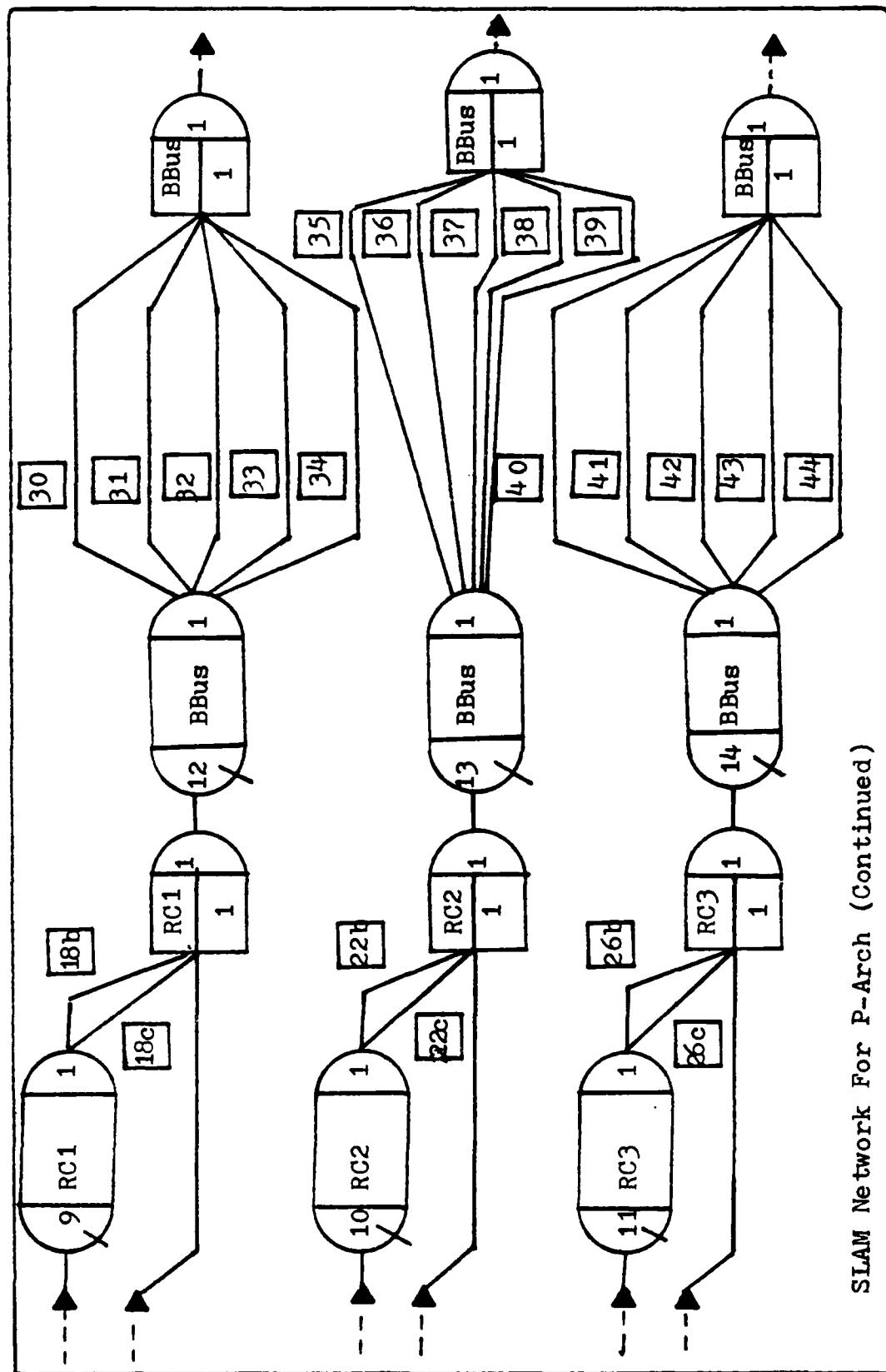


SLAM Network For P-Arch

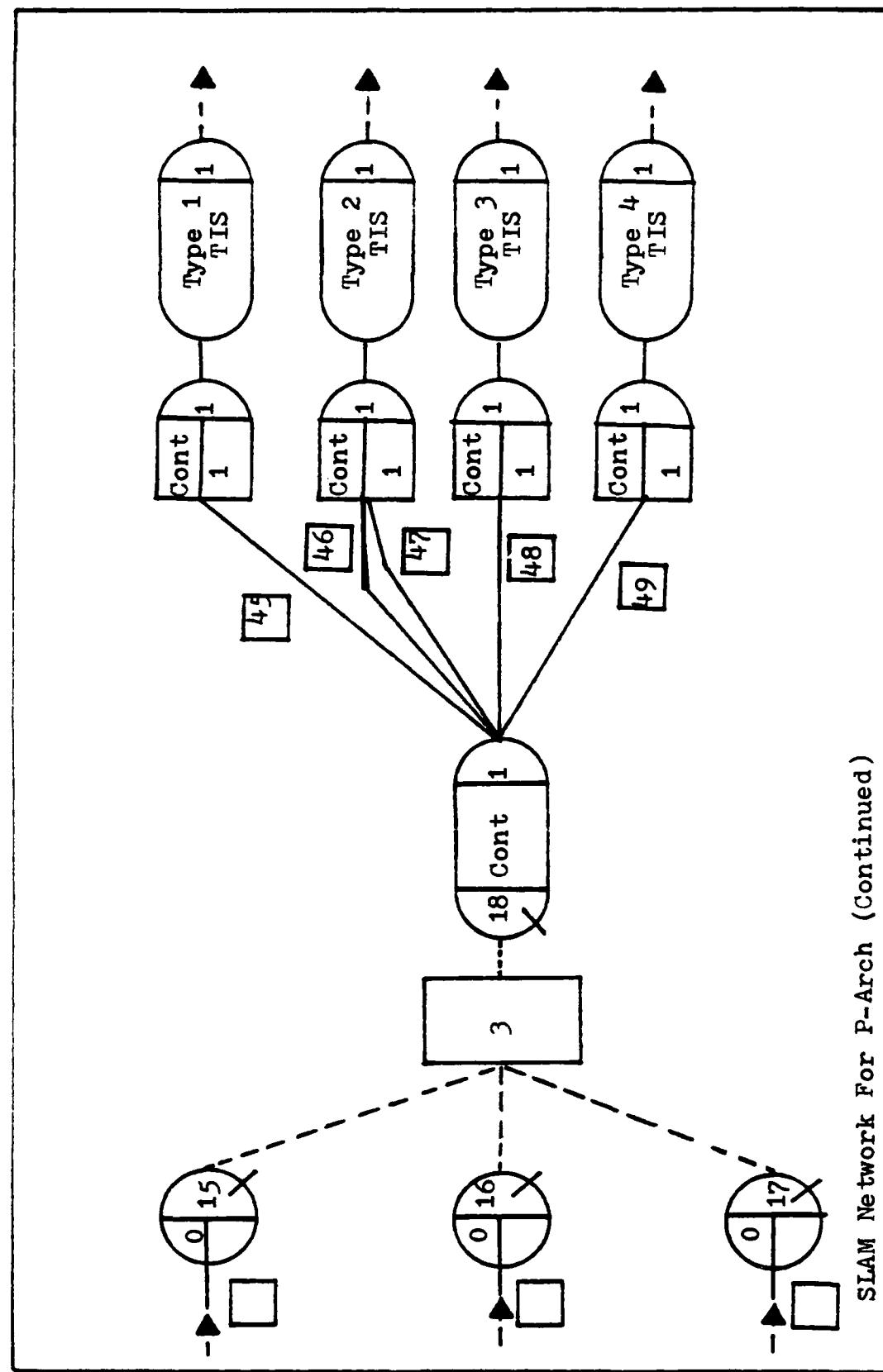




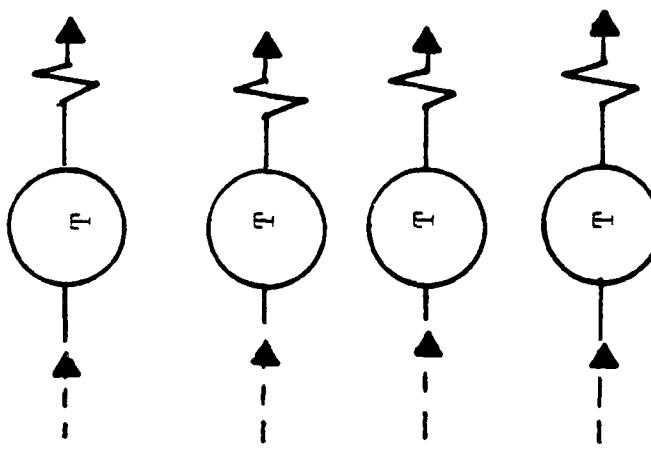
SLAM Network For P-Arch (Continued)



SIAM Network For P-Arch (Continued)



SIAM Network For P-Arch (Continued)



P-Arch Simulation Model Attributes

<u>Attribute</u>	<u>Description</u>
1	Mark Time
2	Query Type
3	Query Identifier
4	Resp1
5	Resp2
6	R1
7	R2
xx(1)	n

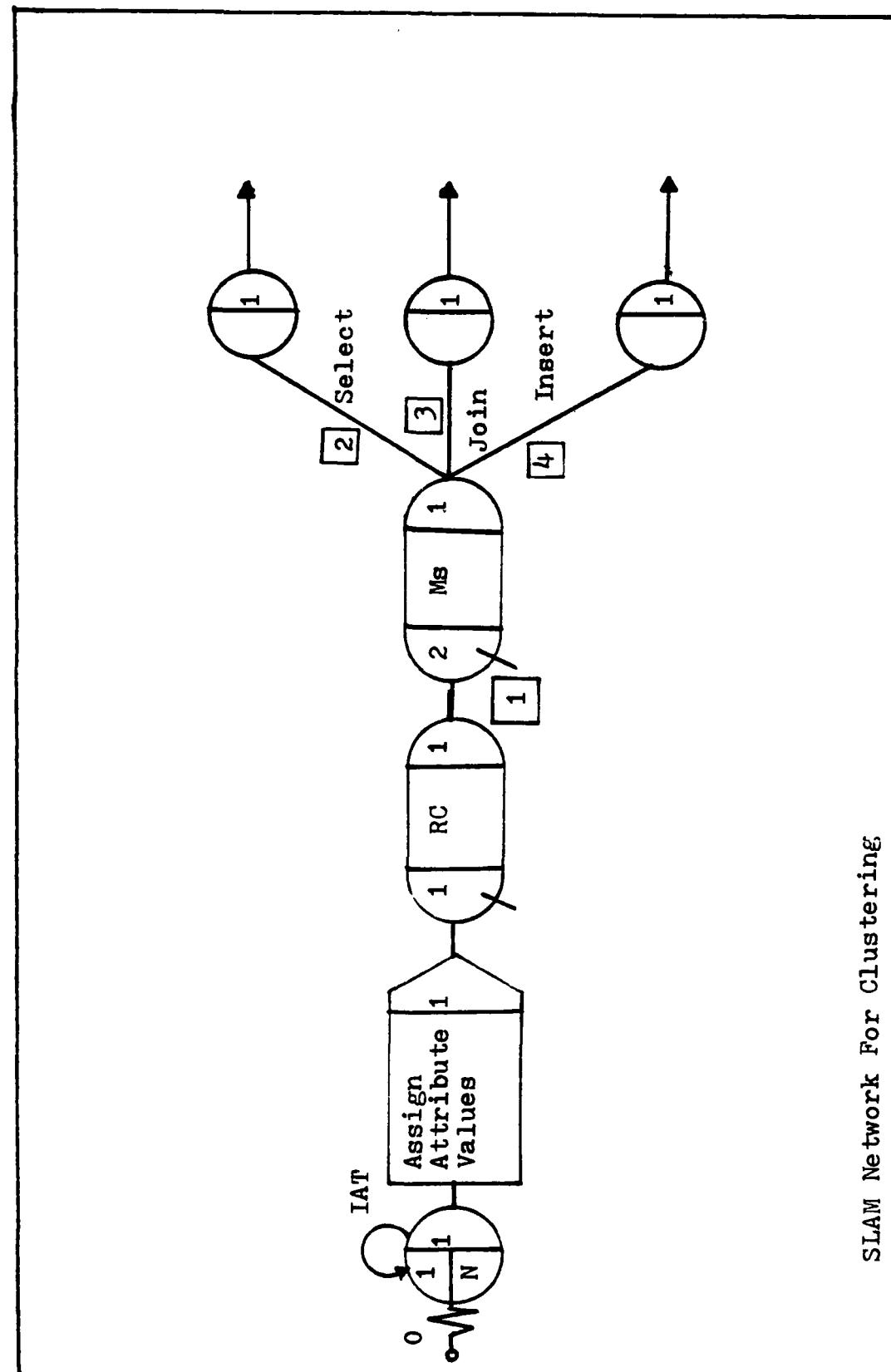
Activities In P-Arch SLAM Simulation Model

<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	Tparse
2	Tmtrans
3,4,5	0
6,10,14	$A(6)*Ta/xx(1)+A(4)*Tr/xx(1)$ when $A(2)=1$
7,11,15	$A(6)*Ta/xx(1)+A(7)*Ta/xx(1)+$ $A(6)*Tr/xx(1)$ when $A(2)=2$
8,12,16	$A(6)*Ta/xx(1)+A(7)*Ta/xx(1)+$ $A(6)*Tr/xx(1)+A(7)*Tr/xx(1)$ when $A(2)=3$
9,13,17	$A(6)*Ta/xx(1)+A(4)*Tr/xx(1)$ when $A(2)=4$
18,22,26	$A(2) = 2$
19,23,27	$A(2) = 1$
20,24,28	$A(2) = 3$
21,25,29	$A(2) = 4$
18a,22a,16a	$A(7)*Trtrans/xx(1)$
18b,22b,26b	$A(4)/xx(1)*A(5)*Tp$ with prob of .50
18c,22c,26c	$A(4)/xx(1)*Tp$ with prob of .50
30,36,40	$A(4)*Trtrans/xx(1)$ when $A(2) = 1$
31,36,41	$A(4)/xx(1)*A(5)*Trtrans$ with prob of .50
32,37,42	$A(4)*Trtrans/xx(1)$ with prob of .50
33,38,43	$A(6)*Trtrans/xx(1)+A(7)*$ $Trtrans/xx(1)$ when $A(2)=3$
34,39,44	Trtrans when $A(2)=4$
45	$A(4)*Tp$ when $A(2)=1$
46	$A(4)*A(5)*Tp$ with prob of .50

47 A(4)*Tp with prob of .50
48 A(6)*Tp+A(7)*Tp when A(2)=3
49 0 when A(2)=4

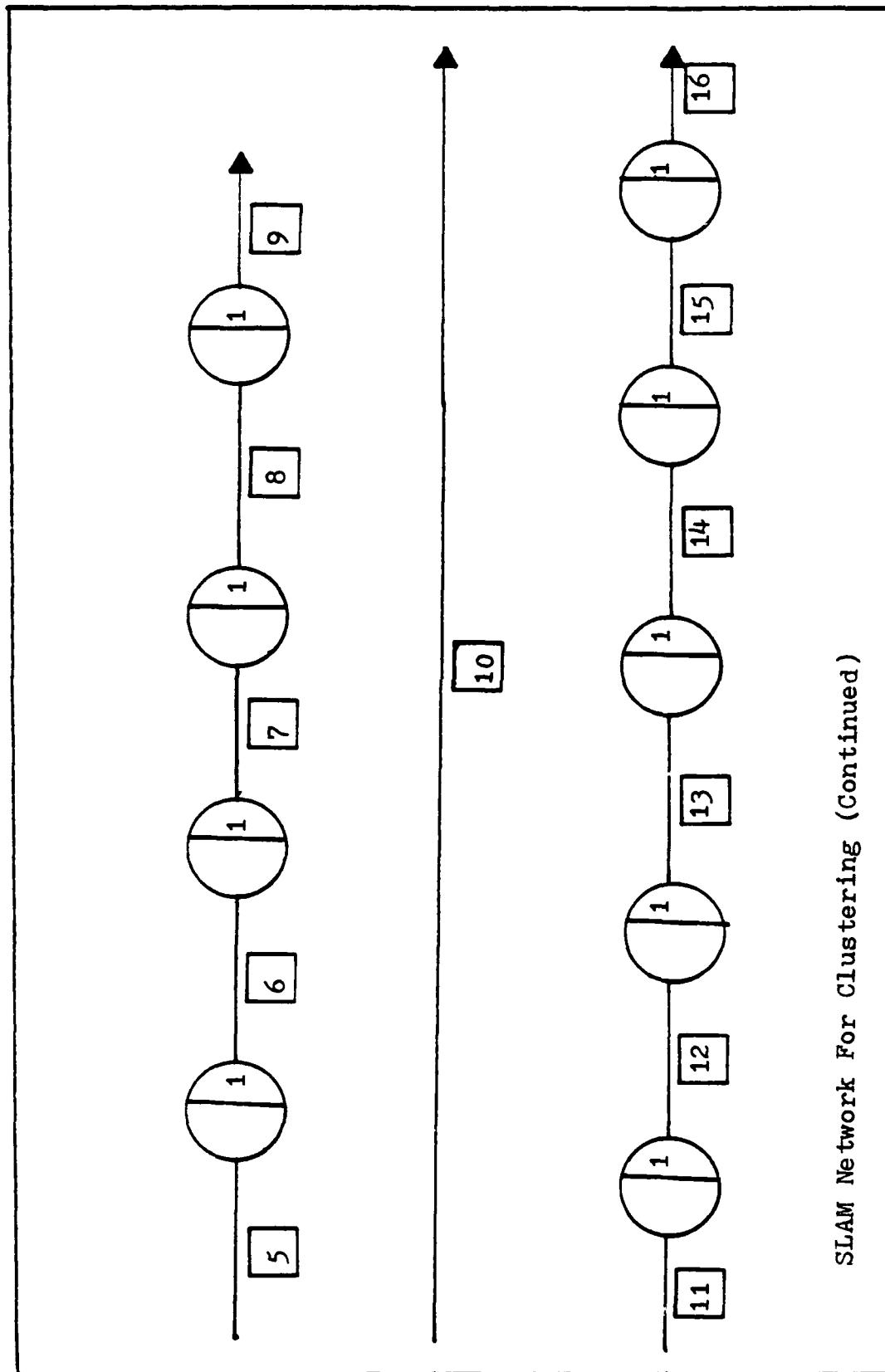
APPENDIX D

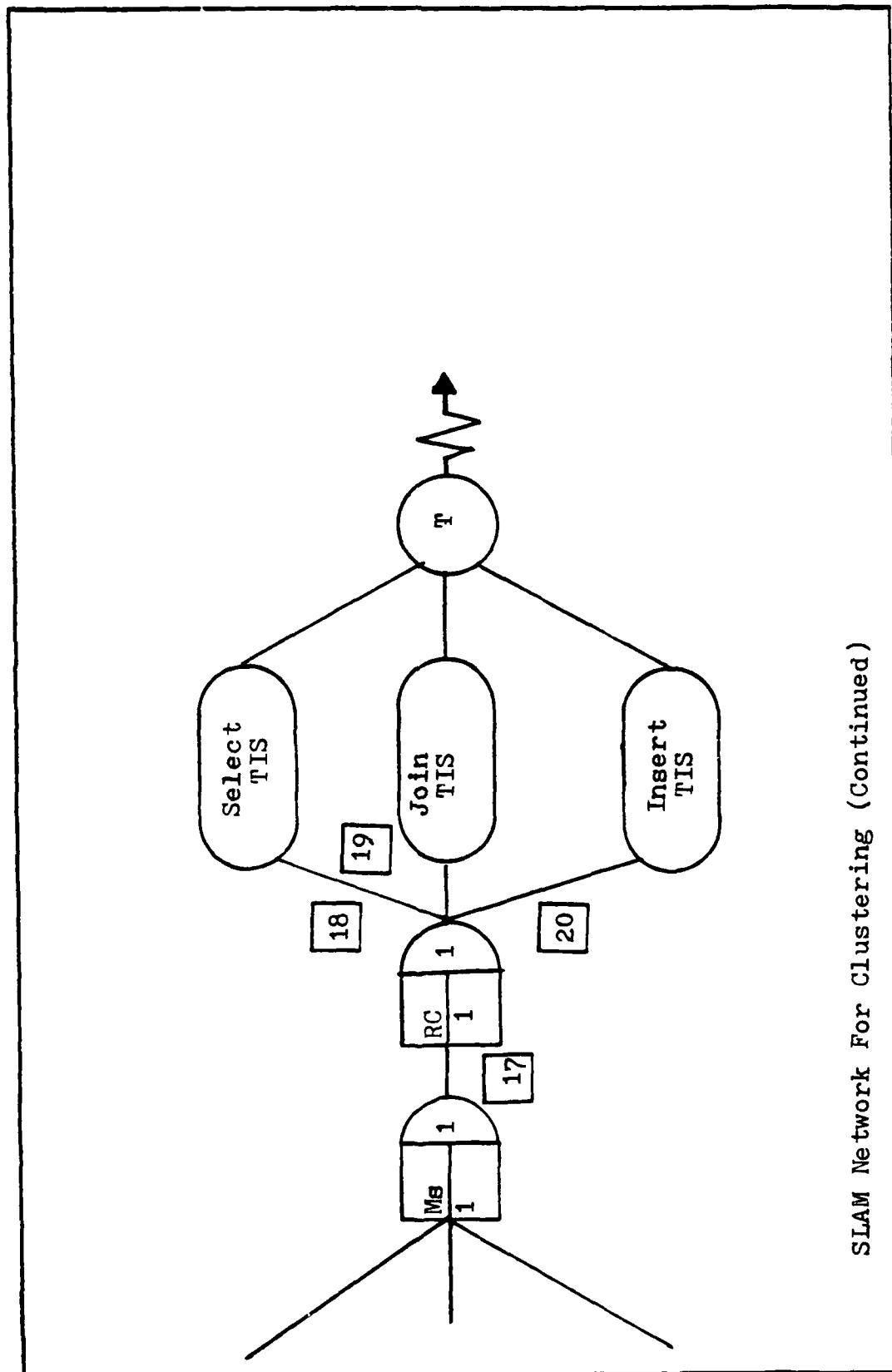
SLAM NETWORKS FOR DATA ACCESS ANALYSIS



SLAM Network For Clustering

SLAM Network For Clustering (Continued)



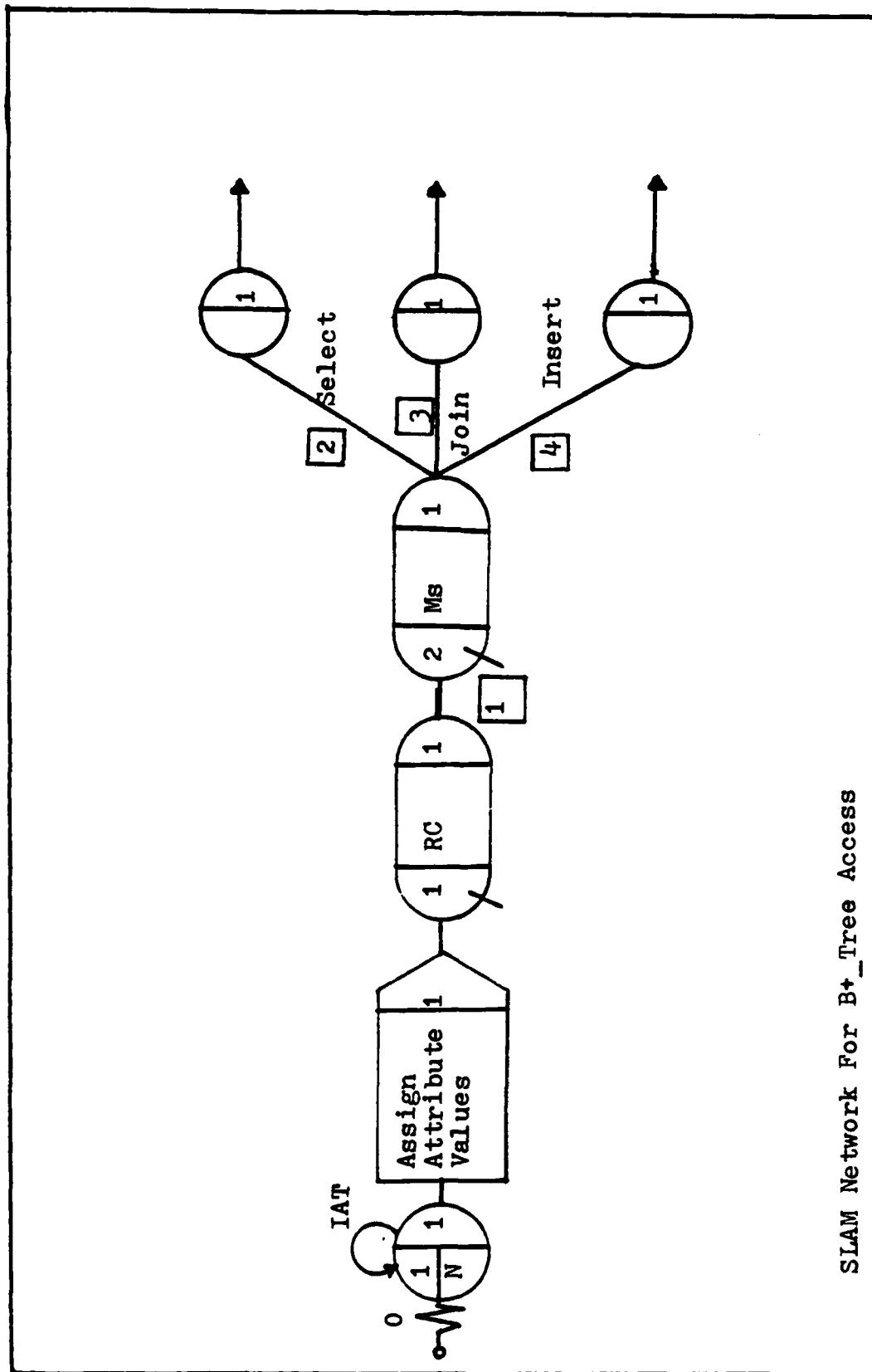


Cluster Simulation Model Attributes

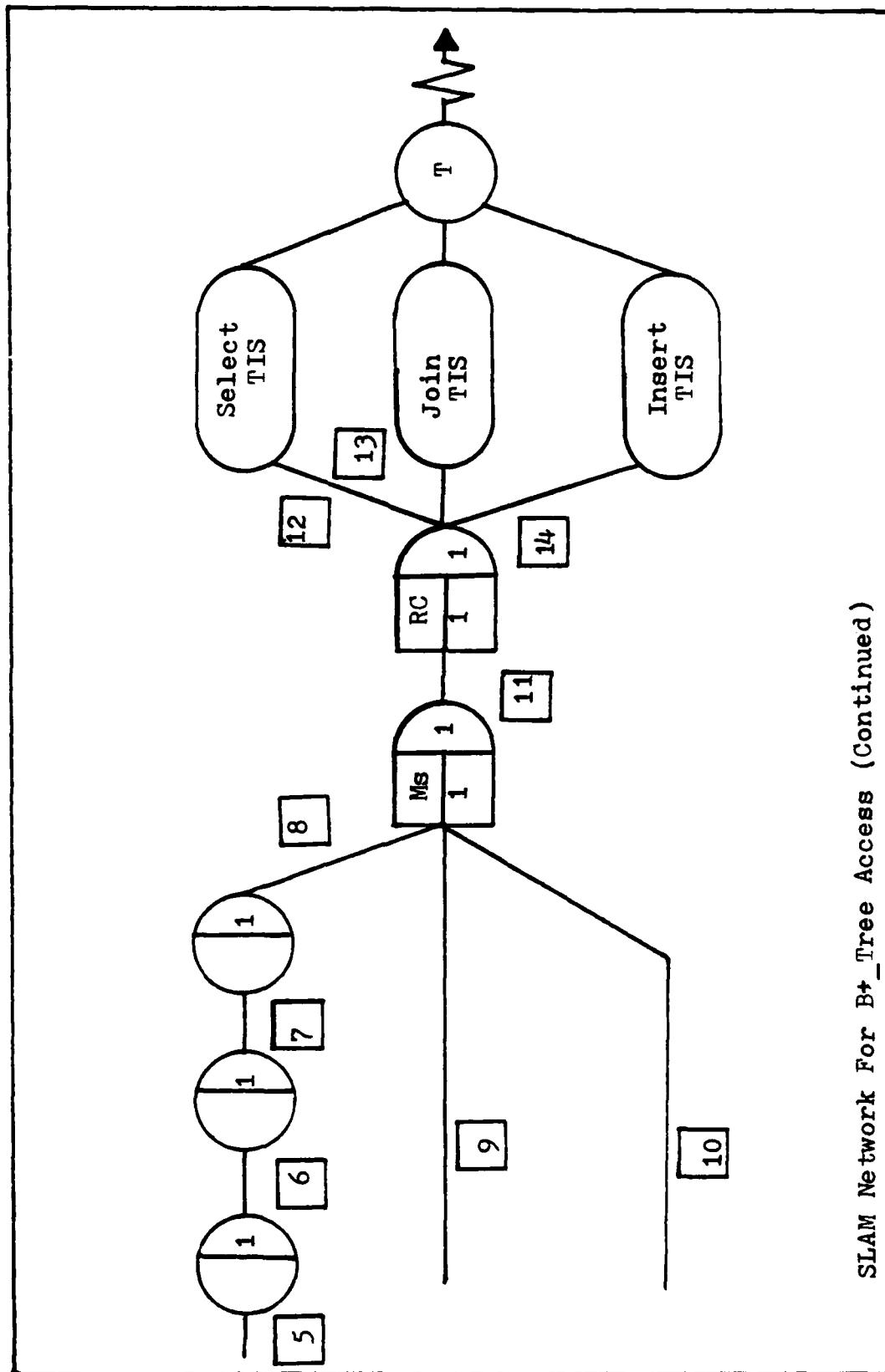
<u>Attribute</u>	<u>Description</u>
1	Mark Time
2	Query Type
3	r
4	a
5	da
6	c
7	p
8	Peq
9	Pda
10	d
11	cl
12	crec
13	dset
14	pdset
15	qclset
16	Pnc
17	1-Pnc
18	1-Peq
19	Number dgroups

Activities In Cluster SLAM Simulation Model

<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	0
2	$A(2) = 1$
3	$A(2) = 2$
4	$A(2) = 3$
5	$A(10) / 2 * Tdt * A(8) * A(6) * A(7)$
6	$A(10) * Tdt * A(18) * A(6) * A(7)$
7	$A(6) * A(19) * Tcc$
8	$A(11) * Tdset + A(11) * A(6) * A(19) * Tcomp$
9	$A(15) * A(12) * Taaddr$
10	$2 * A(11) * A(12) * Taaddr$
11	$A(5) * A(10) / 2 * Tdt$
12	$A(17) * A(11) / 2 * A(13) * TdDid$
13	$A(17) * 2 * Taaddr$
14	$A(16) * A(11) * A(13) * TdDid$
15	$A(16) * A(13) * TdDid$
16	$A(16) * Taaddr$
17	0
18	$A(2) = 1$
19	$A(2) = 2$
20	$A(2) = 3$



SLAM Network For B+ Tree Access



B+_tree Simulation Model Attributes

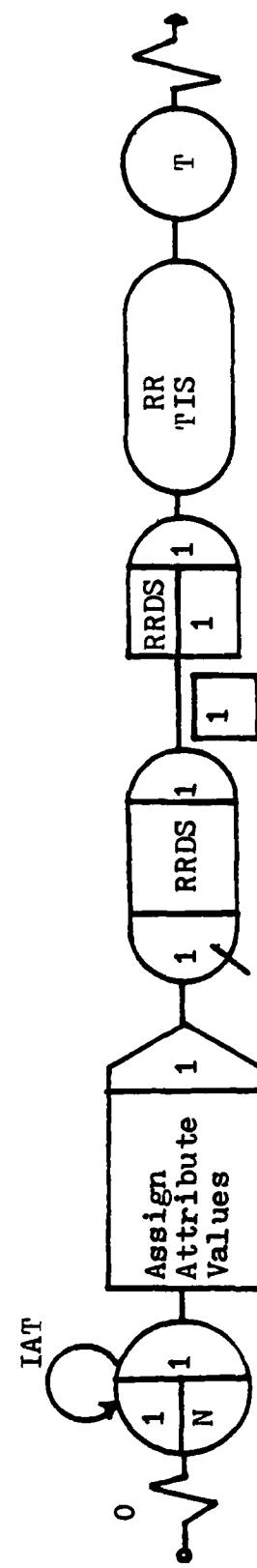
<u>Attribute</u>	<u>Description</u>
1	Mark Time
2	Query Type
3	r
4	a
5	da
6	c
7	p
8	Peq
9	Pda
10	h
11	k
12	n
13	nl
14	Presp
15	Read/Write for insert
18	1 - Peq

Activities In B+_tree SLAM Simulation Model

<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	0
2	$A(2) = 1$
3	$A(2) = 2$
4	$A(2) = 3$
5	$A(6) * A(7) * A(8) * A(10) * Tb$
6	$A(6) * A(7) * A(18) * A(10) * Tb$
7	$A(6) * A(7) * A(18) * A(13) / 2 * Tb$
8	$A(6) * A(7) * A(14) * Taddr$
9	$2 * A(3) * Taddr$
10	$A(5) * A(15) * Tb + Taddr$
11	0
12	$A(2) = 1$
13	$A(2) = 2$
14	$A(2) = 3$

APPENDIX E

SLAM NETWORKS FOR DATA PLACEMENT ANALYSIS



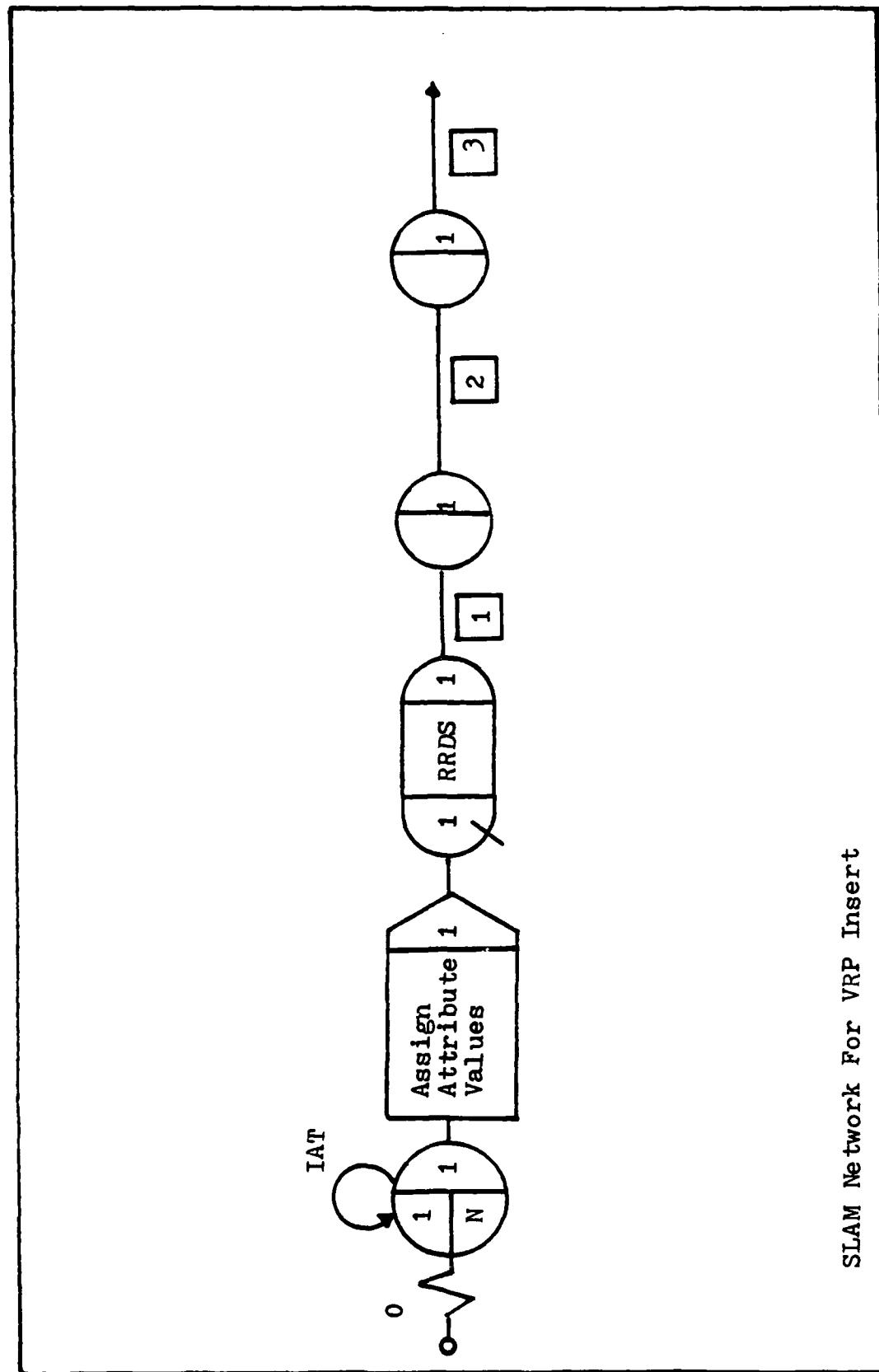
SLAM Network For RR Insert

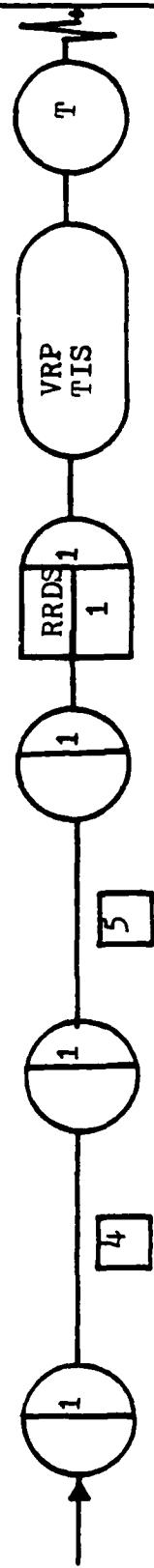
RR Insert Simulation Model Attributes

<u>Attribute</u>	<u>Description</u>
1	Mark Time
3	r
4	a
5	da
9	Pda
11	k
12	n
15	Read/Write for insert
18	1 - Peq

Activities In RR Insert SLAM Simulation Model

<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	$A(5) * A(15) * Tb + Taddr$





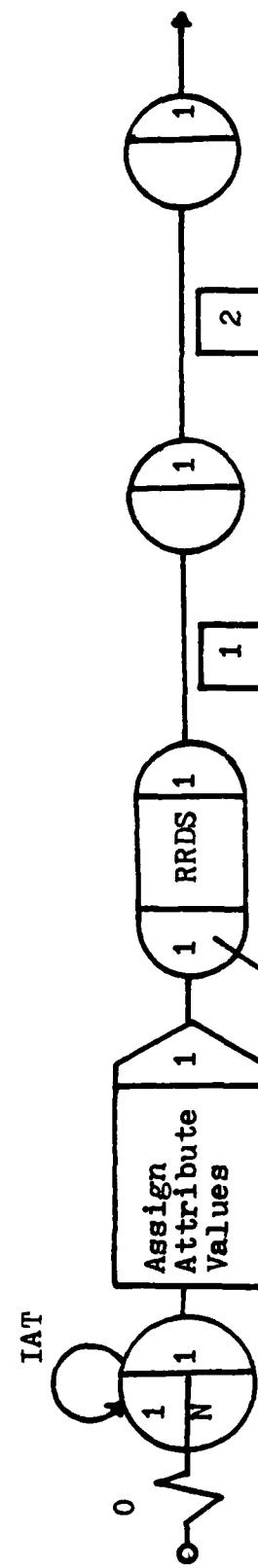
SLAM Network For VRP Insert (Continued)

VRP Insert Simulation Model Attributes

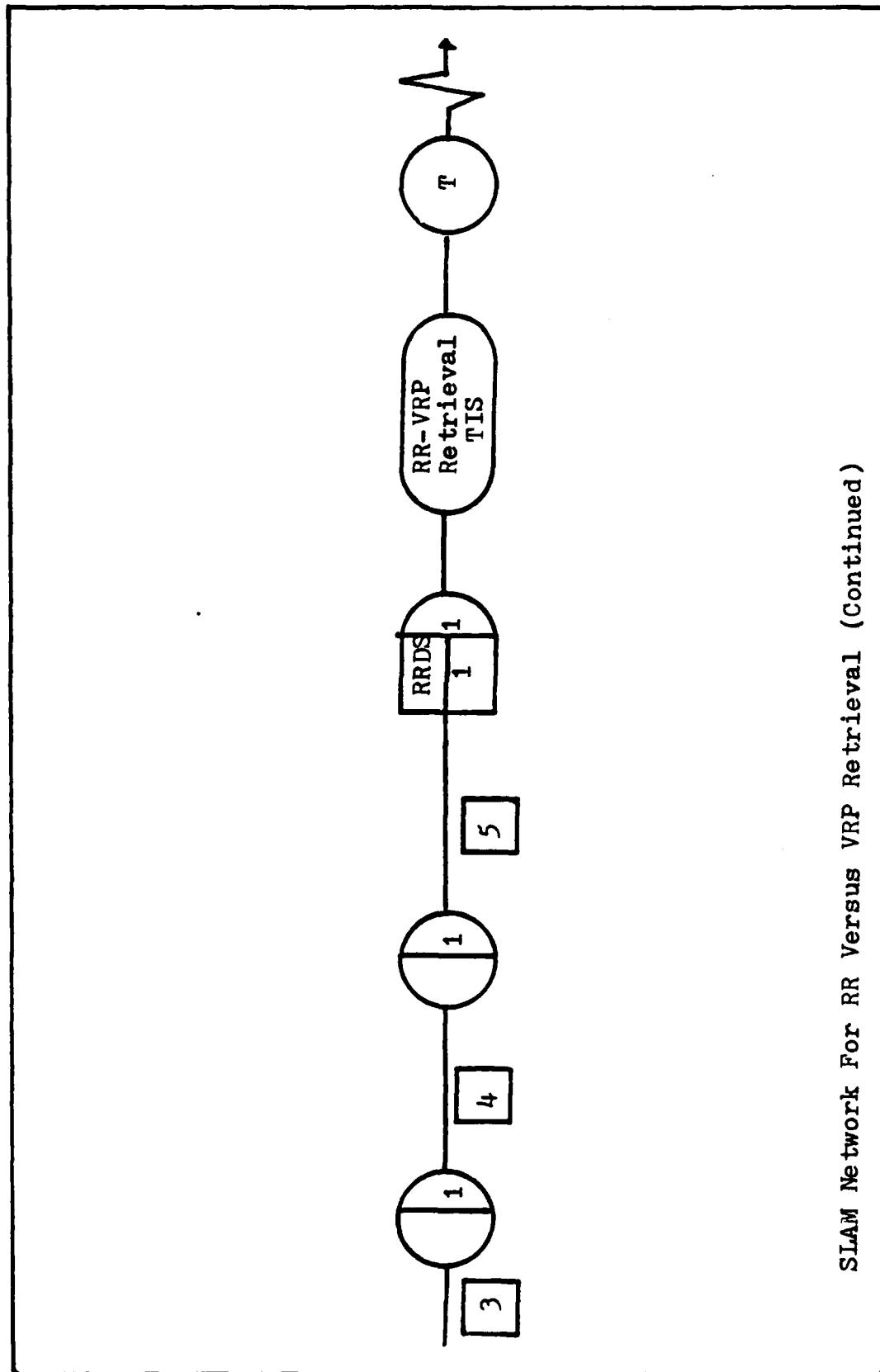
Attribute	Description
1	Mark Time
3	r
4	a
5	da
6	pa
7	PT
8	Ppa
9	Pda
10	DT
11	k
12	n
13	dset
15	Read/Write for Insert
16	Pnp
17	1-Pnp
18	1-Peq

Activities In VRP Insert SLAM Simulation Model

<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	$A(6) * A(10) / 2 * Tdt$
2	$A(17) * A(7) / 2 * A(13) * Tdid$
3	$A(16) * A(7) * A(13) * Tdid$
4	$A(16) * A(13) * Tdid$
5	$A(5) * A(15) * Tb + Taddr$



SLAM Network For RR Versus VRP Retrieval



RR versus VRP Retrieval Simulation Model Attributes

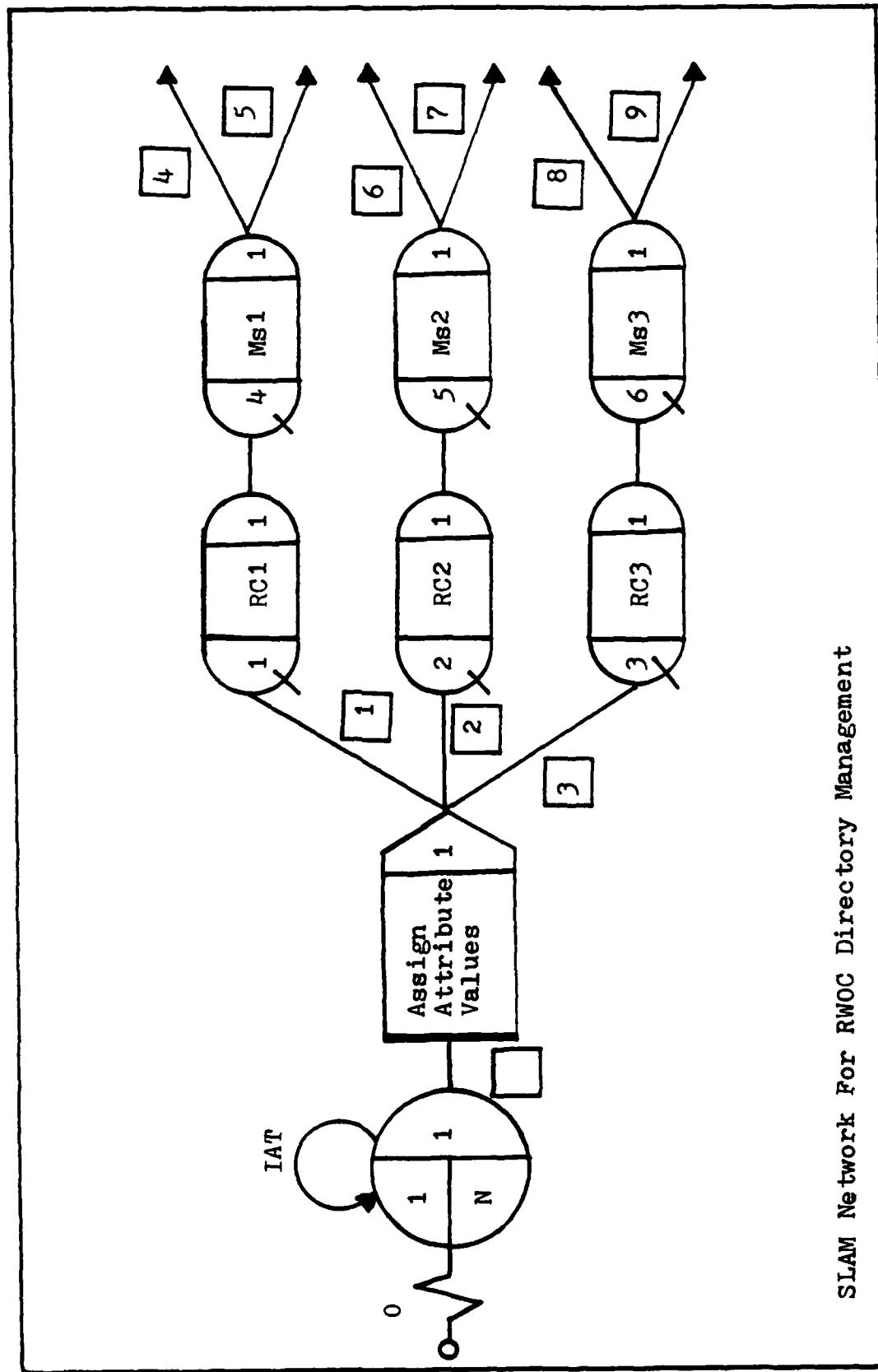
<u>Attribute</u>	<u>Description</u>
1	Mark Time
3	r
4	a
5	da
6	c
7	p
8	Peq
9	Pda
10	h
11	k
12	n
13	nl
14	Presp
15	Number of RCs in RRDS
16	Percent of system in use
17	Degree of parallelism
18	1-Peq
19	PT
20	Query response set size

Activities in RR versus VRP Retrieval Simulation Model

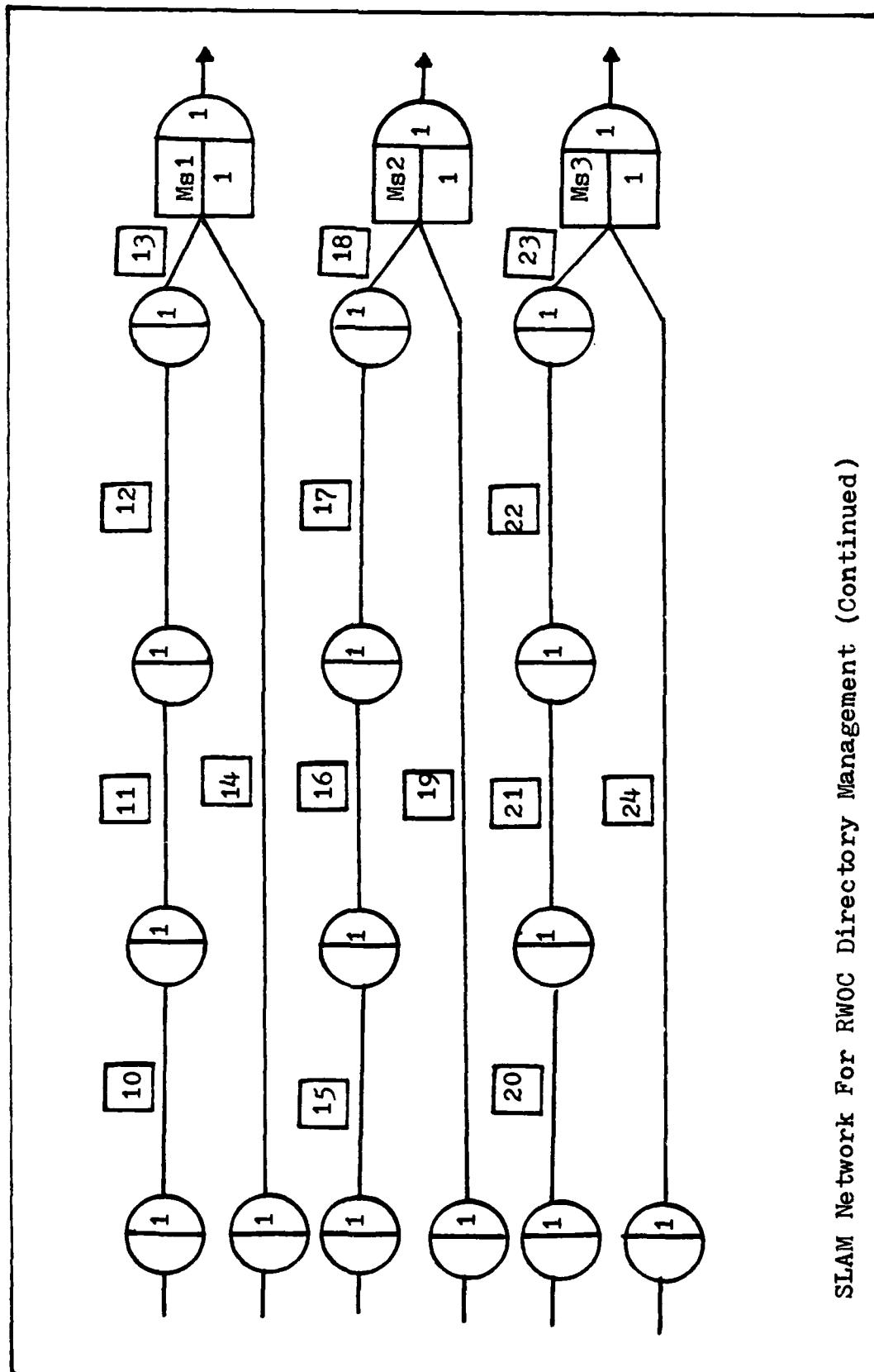
<u>Activity</u>	<u>Duration or Branching Condition</u>
1	$A(6) * A(7) * A(8) * A(10) * Tb$
2	$A(6) * A(7) * A(18) * A(10) * Tb$
3	$A(6) * A(7) * A(18) * A(13) / 2 * Tb$
4	$A(6) * A(7) * A(14) * Taddr / A(17)$
5	$A(20) * Tread / A(17)$ (Tread = 3.51ms for a 100-byte record)

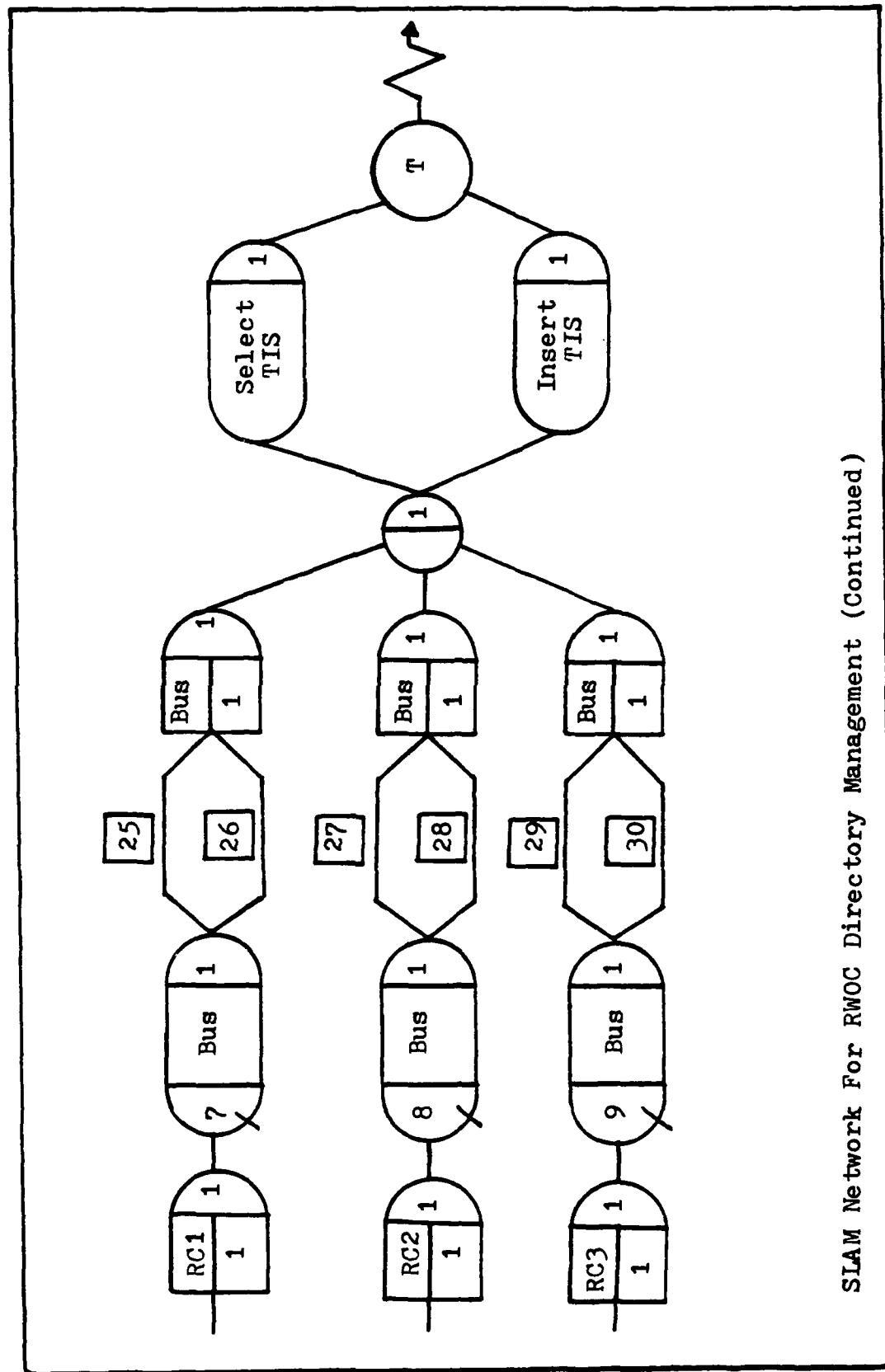
APPENDIX F

SLAM NETWORKS FOR DIRECTORY MANAGEMENT ANALYSIS



SLAM Network For RWC Directory Management



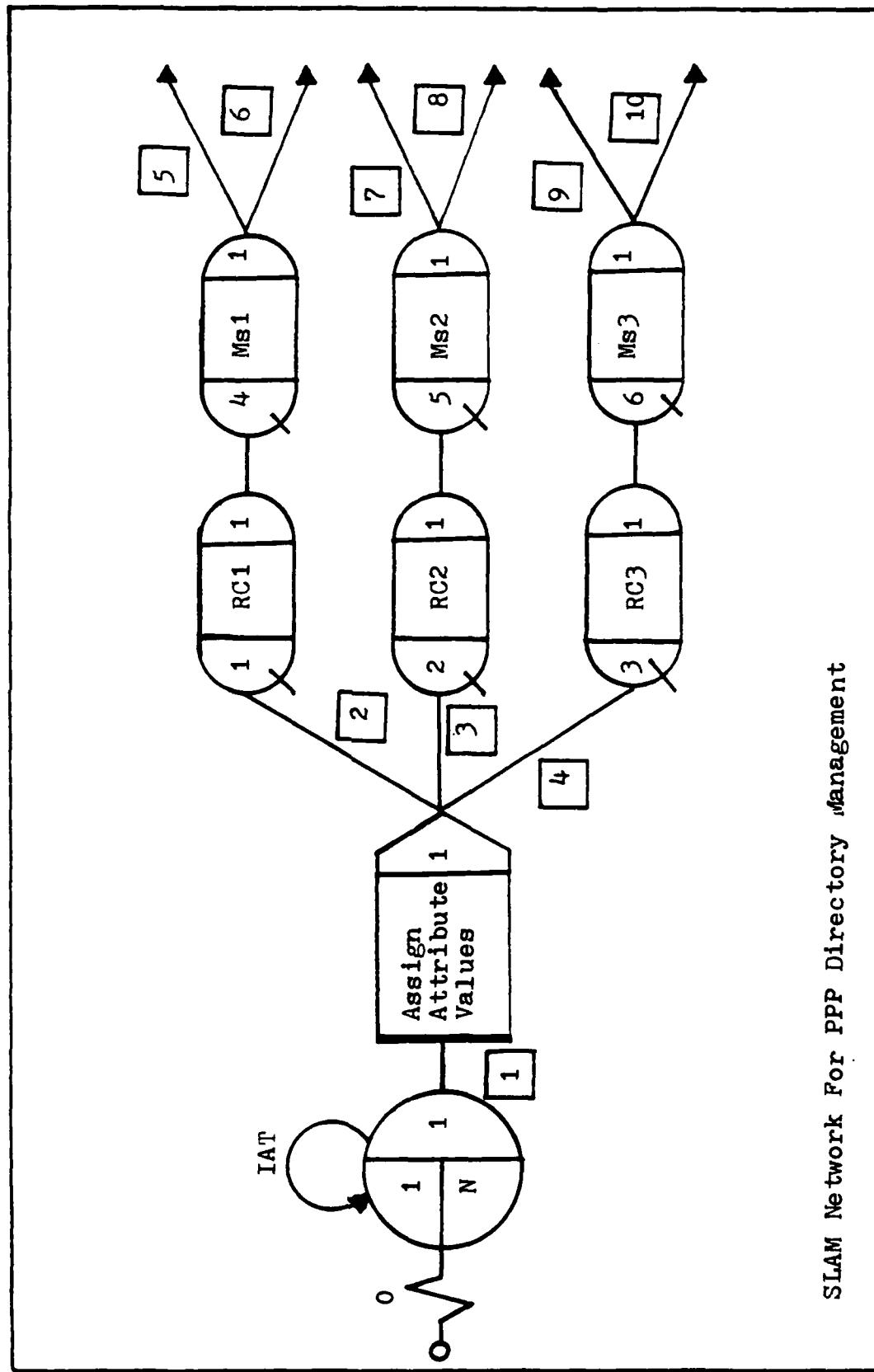


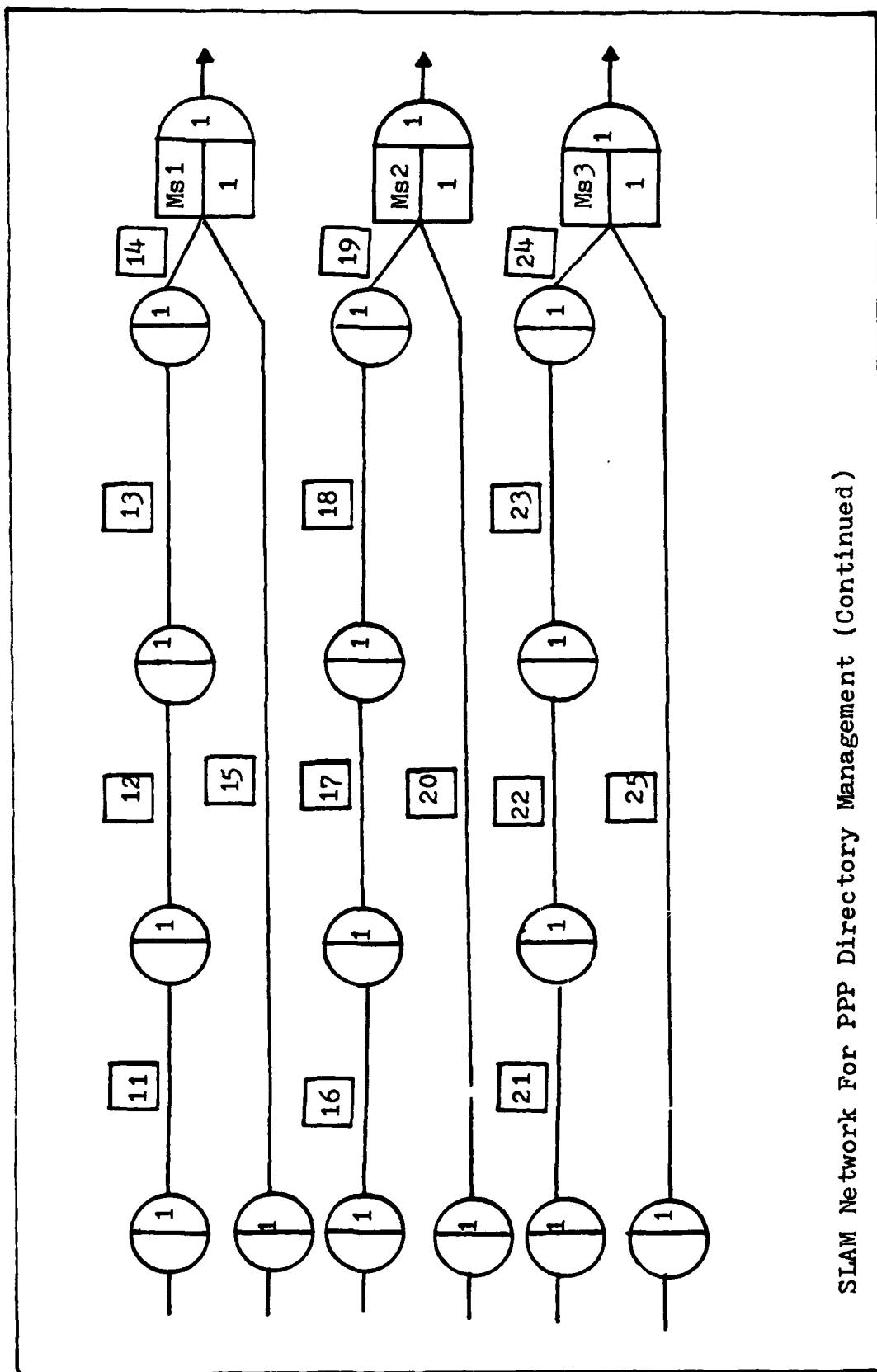
RWOC Simulation Model Attributes

Attribute	Description
1	Mark Time
2	Query Type
3	r
4	a
5	da
6	c
7	p
8	Peq
9	Pda
10	h
11	k
12	n
13	nl
14	Presp
15	Read/Write for insert
16	Query ID
18	1 - Peq
19	Tadtrans
20	Bus Address Multiplier
21	RCid of Directory Processor

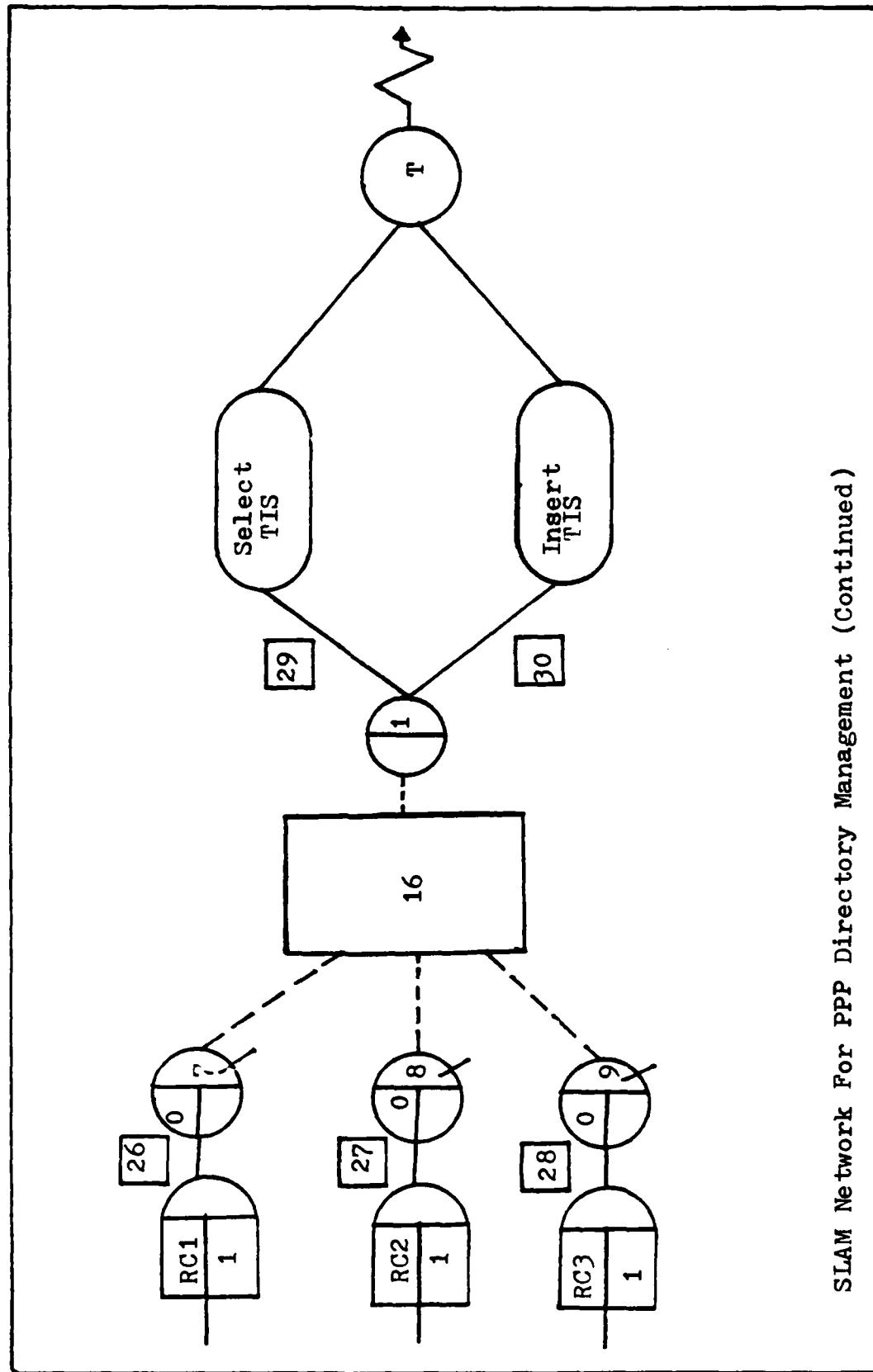
Activities In RWOC SLAM Simulation Model

<u>Activity</u>	<u>Duration Or Branching Condition</u>
1	$A(21) = 0$
2	$A(21) = 1$
3	$A(21) = 2$
4,6,8	$A(2) = 1$
5,7,9	$A(2) = 3$
10,15,20	$A(6)*A(7)*A(8)*A(10)*Tb$
11,16,21	$A(6)*A(7)*A(18)*A(10)*Tb$
12,17,22	$A(6)*A(7)*A(18)*A(13)/2*Tb$
13,18,23	$A(6)*A(7)*A(14)*Taddr$
14,19,24	$A(5)*A(15)*Tb+Taddr$
25,27,29	$A(6)*A(7)*A(14)*A(19)*A(20)$ when $A(2) = 1$
26,28,30	$A(19)*A(20)$ when $A(2) = 3$





SLAM Network For PPP Directory Management (Continued)



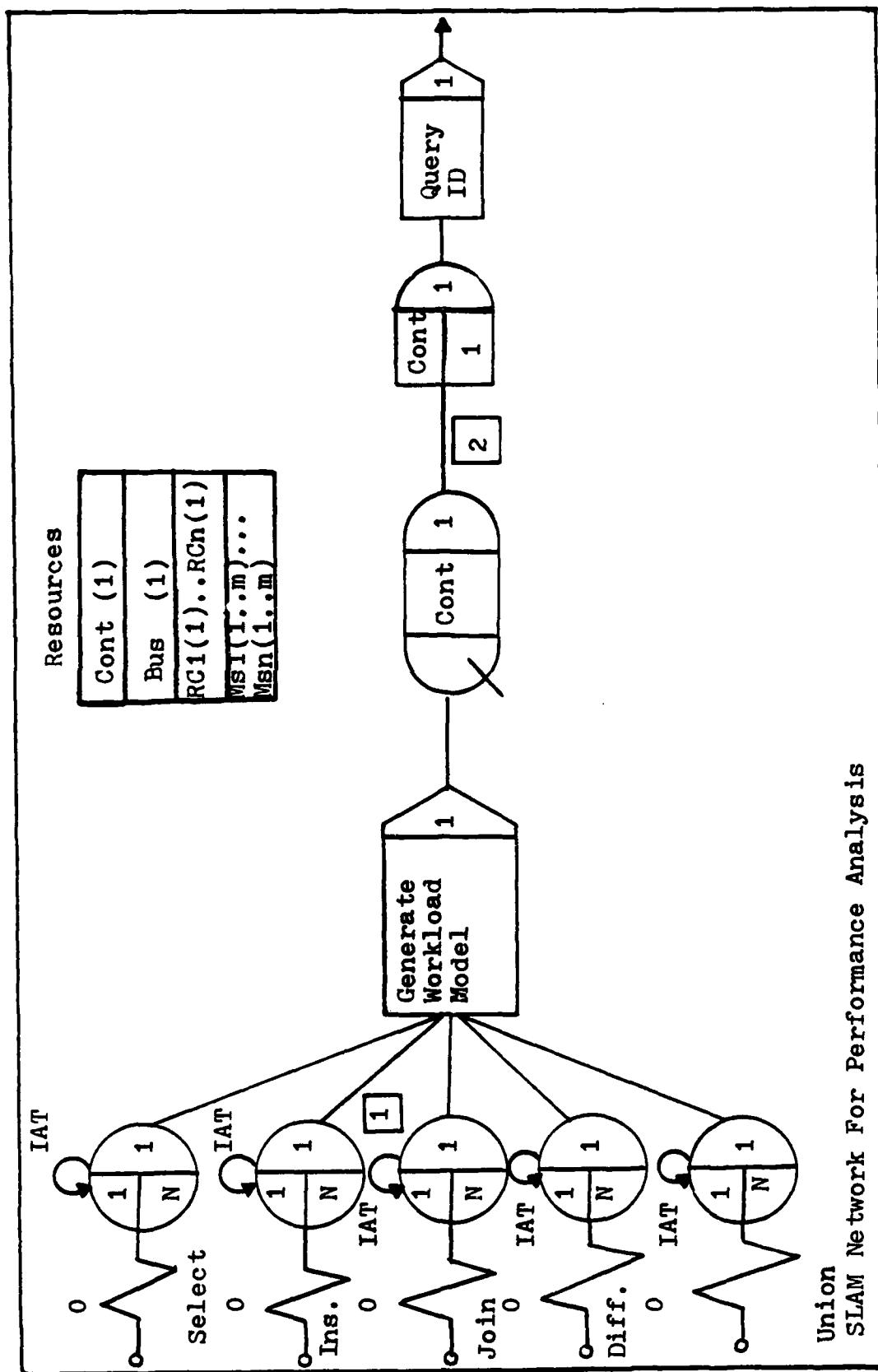
PPP Simulation Model Attributes

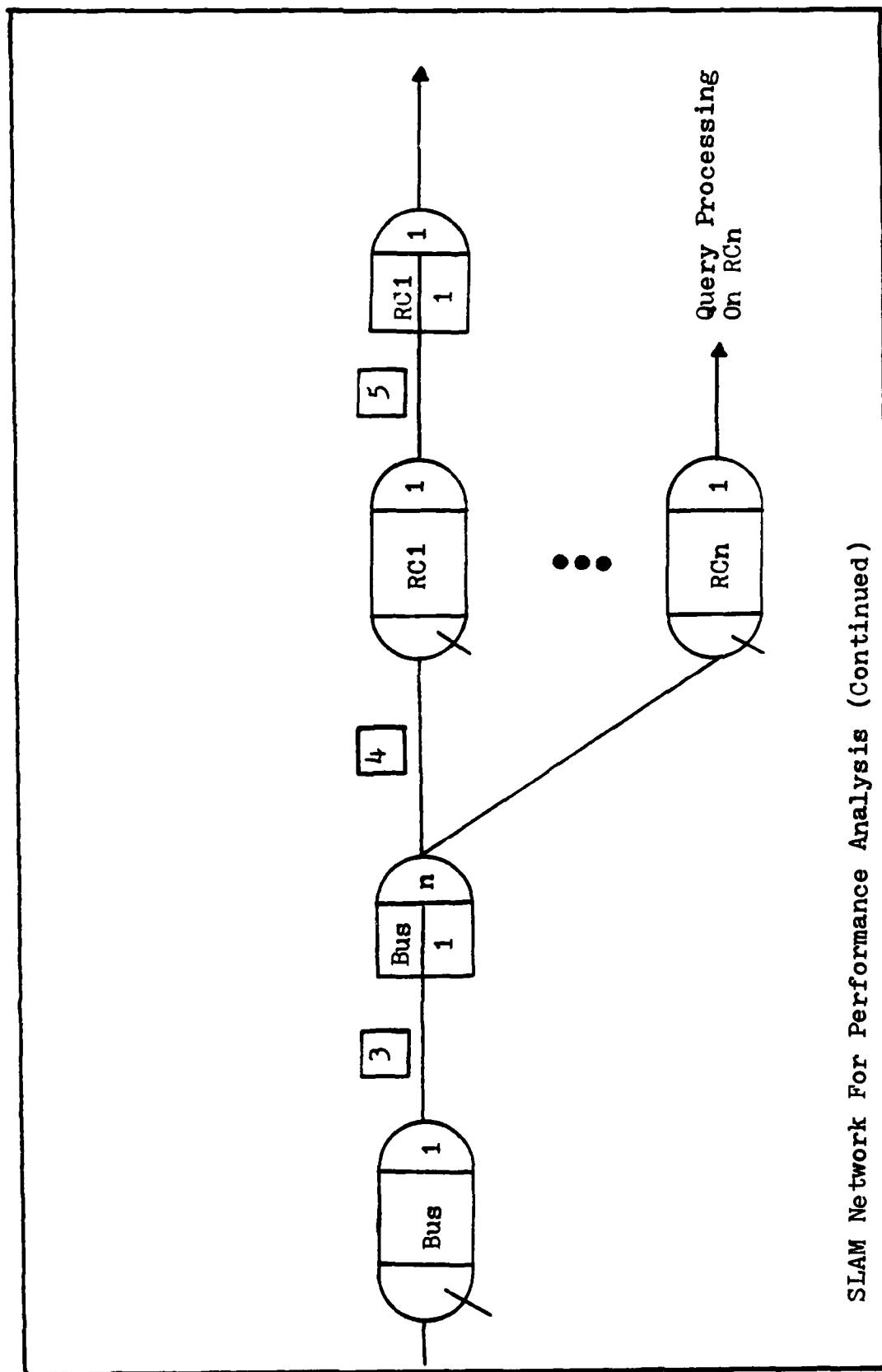
<u>Attribute</u>	<u>Description</u>
1	Mark Time
2	Query Type
3	r
4	a
5	da
6	c
7	p
8	Peq
9	Pda
10	h
11	k
12	n
13	nl
14	Presp
15	Read/Write for Insert
16	Query ID
18	1-Peq

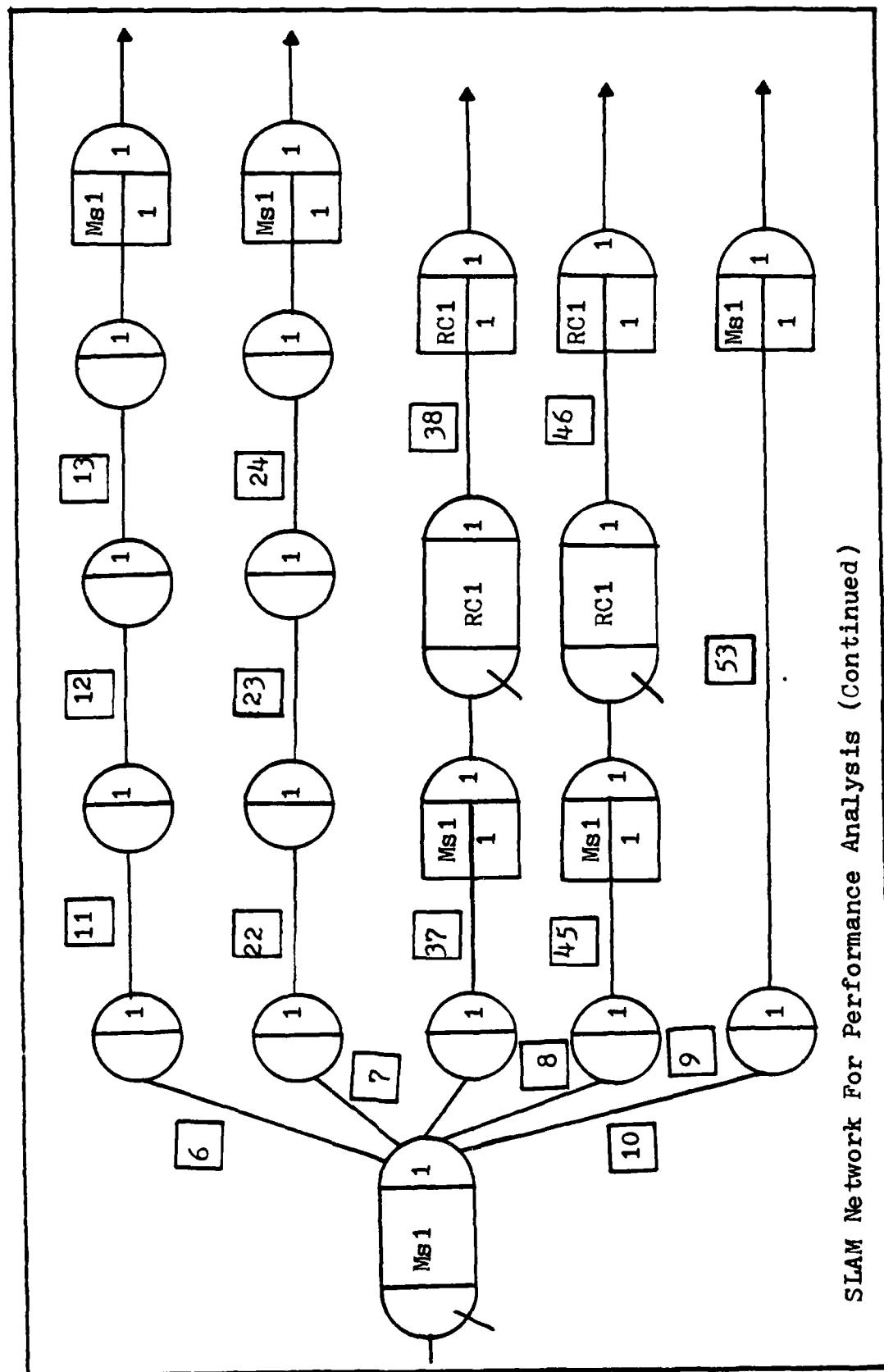
Activities In PPP SLAM Simulation Model

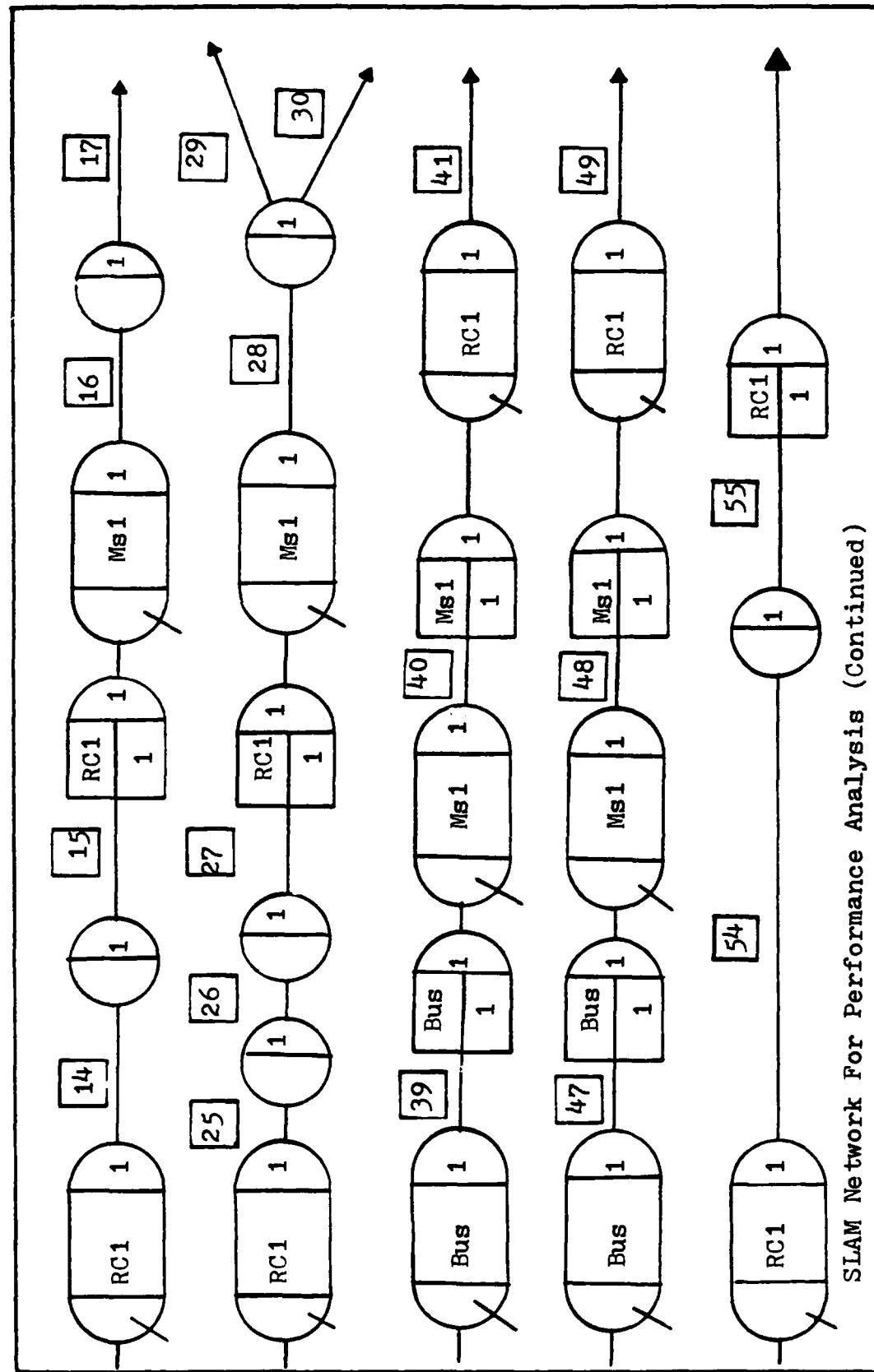
Activity	Duration Or Branching Condition
1	0
2,3,4	0
5,7,9	$A(2) = 1$
6,8,10	$A(2) = 3$
11,16,21	$A(6) * A(7) * A(8) * A(10) * Tb$
12,17,22	$A(6) * A(7) * A(18) * A(10) * Tb$
13,18,23	$A(6) * A(7) * A(18) * A(13) / 2 * Tb$
14,19,24	$A(6) * A(7) * A(14) * Taddr$
15,20,25	$A(5) * A(15) * Tb + Taddr$
26,27,28	0
29	$A(2) = 1$
30	$A(2) = 3$

APPENDIX G
SLAM NETWORKS FOR PERFORMANCE ANALYSIS

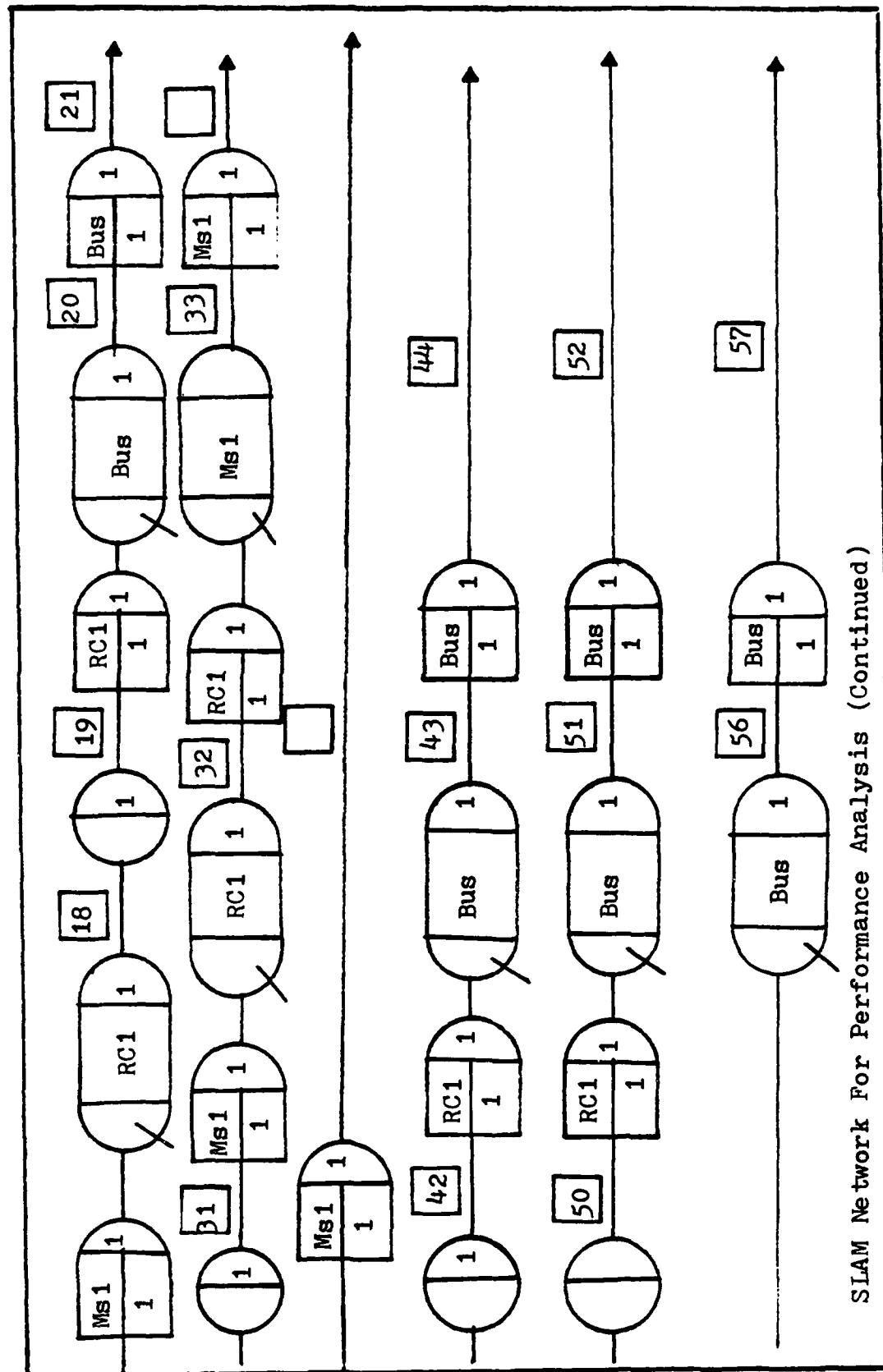




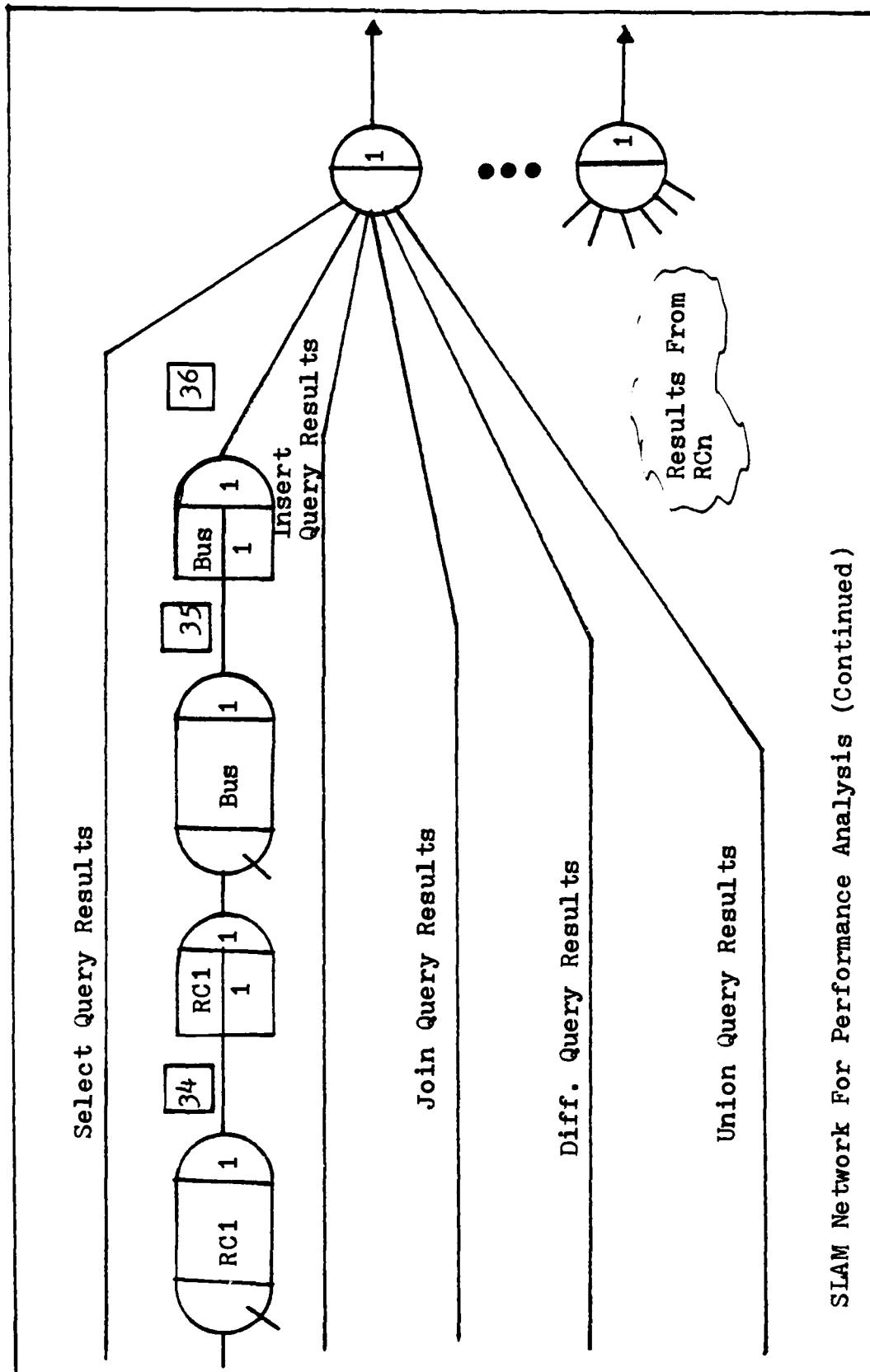




SIAM Network For Performance Analysis (Continued)

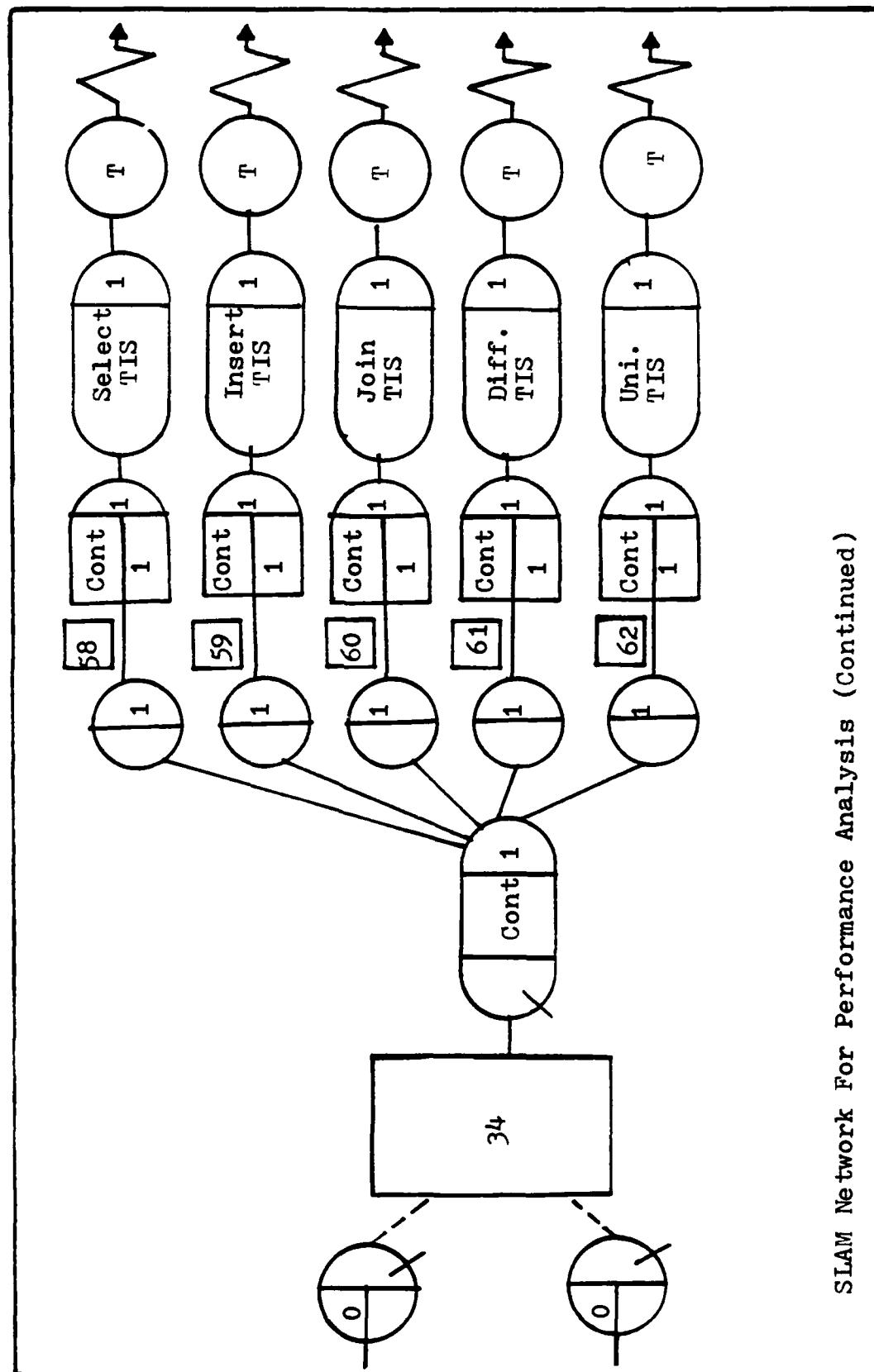


SLAM Network For Performance Analysis (Continued)



SLAM Network For Performance Analysis (Continued)

SLAM Network For Performance Analysis (Continued)



Perfomance Analysis Simulation Model Attributes

Attribute	Description
1	Mark Time
2	Query Type
3	r1
4	a
5	da
6	c
7	p
8	Peq
9	Pda
10	h
11	k
12	n
13	nl
14	Presp
15	Read/Write for insert
18	1 - Peq
20	pa
21	PT
22	Ppa
23	DT
24	dset
25	Pnp
26	1 - Pnp
27	RCid for insertion
30	r2
31	MIN(r1,r2)
32	MAX(r1,r2)
33	Number of RCs
34	Query ID
35	%Result

Activities In Performance Analysis SLAM Simulation Model

Activity	Duration Or Branching Condition
1	0
2	Tparsē
3	Tmtrans
4	0
5	0
6	A(2) = 1
7	A(2) = 2
8	A(2) = 3
9	A(2) = 4
10	A(2) = 5
11	A(6)*A(7)*A(8)*A(10)*Tb
12	A(6)*A(7)*A(18)*A(10)*Tb
13	A(6)*A(7)*A(18)*A(13)/2*Tb
14	A(6)*A(7)*A(8)*A(10)*Tbproc
15	A(6)*A(7)*A(18)*A(10)*Tbproc
16	A(6)*A(7)*A(14)*Taddr
17	A(6)*A(7)*A(14)*Tread
18	A(6)*A(7)*A(14)*Tproc
19	Tgen
20	A(6)*A(7)*A(14)*A(35)*Trectrans+Tmtrans
21	0
22	A(20)*A(23)/2*Tdt
23	A(26)*A(21)/2*A(24)*Tdid
24	A(25)*A(21)*A(24)*Tdid

25 $A(20)*A(23)/2*Ttproc$
26 $A(26)*A(21)/2*A(24)*Tdidproc$
27 $A(25)*A(21)*A(24)*Tdidproc$
28 $A(25)*A(24)*Tdid$
29 $A(27) = RCid$ of this RC
 (insertion at this RC)
30 $A(27) <> RCid$ of this RC
 (insertion not at this RC)
31 $A(5)*A(15)*Tb+Taddr$
32 $A(5)*A(15)*Tbproc$
33 Twrite (Tread)
34 Tgen
35 Tmtrans
36 0
37 $A(31)/A(33)*Tread$
38 Tgen
39 $A(31)/A(33)*Trectrans+Tmtrans$
40 $A(32)/A(33)*Tread$
41 $A(31)*A(32)/A(33)*Tproc$
42 Tgen
43 $A(31)*A(32)/A(33)*A(35)*Trectrans+Tmtrans$
44 0
45 $A(30)/A(33)*Tread$
46 Tgen
47 $A(30)/A(33)*Trectrans+Tmtrans$
48 $A(3)/A(33)*Tread$
49 $A(3)/A(33)*A(30)*Tproc$
50 Tgen

51 $A(3)*A(35)/A(33)*Trectrans+Tmtrans$
52 0
53 $A(3)/A(33)*Tread+A(30)/A(33)*Tread$
54 $A(3)/A(33)*A(30)/A(33)*Tproc$
55 Tgen
56 $A(3)/A(33)*Trectrans+A(30)/A(33)*A(35)*$
 Trectrans+Tmtrans
57 0
58,59,60,61 0
62 $A(3)*A(30)*A(35)*Tproc$

APPENDIX H

VITA

Captain Jon G. Hanson [REDACTED]

[REDACTED] In 1973 he graduated from Hampton High School in Hampton, Virginia. He attended the United States Air Force Academy from which he received a Bachelor of Science degree in 1977. Following graduation he attended Communication Operations Officer School at Keesler AFB, Mississippi. Between April 1978 and April 1980 he was assigned to the 2192nd Communications Squadron, Air Force Communications Command, at Loring AFB, Maine as Combat Crew Communications Officer for the 42d Bombardment Wing. He then attended the Air Force Institute of Technology from which he received a Master of Science Degree, in Information Systems, in 1981. Between January 1982 and April 1984 he served as Chief, Systems Software Section, for the Computer Assisted Force Management System Division under the Tactical Air Command single manager for data automation, at Langley AFB, Virginia. He entered the University of Central Florida in May 1984.

BIBLIOGRAPHY

Adiba, M., Chupin, F., Demolombe, F., Gardaria, F., LeBihan, J., "Issues in Distributed Database Management Systems - A Technical Overview," Proceedings of the IEEE Computer Society Fourth International Conference on Very Large Databases, September 1978.

Astrahan, M., Blasgen, M., Chamberlin, D., Eswaran, K., Gray, V., Griffith, P., King, W., Lorie, R., McJones, P., Mehl, J., Potzolu, G., Traiger, J., Wade, B., Watson, V., "System R: A Relational Approach to Database Management," ACM Transactions on Database Systems, 1, 3, September 1976.

Auer, H., "RDBM - A Relational Database Machine," Technical Report No. 8005, University of Braunschweig, June 1980.

Baer, Jean L., Computer Systems Architecture, Computer Science Press, Rockville, Maryland, 1980.

Banerjee, J., Baum, R.I., and Hsiao, D.K., "Concepts and Capabilities of a Database Computer," ACM Transactions on Database Systems, 3, 4, December 1978.

Berr, P.B., and Oliver, E., "The Role Of Array Processors In Database Machine Architecture," IEEE Computer, Vol. 12, No. 3, 1979.

Birtwhistle, G.M., Dahl O., Myhrhaug, B. and Nygaard, K. Simula Begin, Auerbach, 1973.

Boral, H., et. al., "Parallel Algorithms for the Execution of Relational Database Operations," Computer Sciences Technical Report No. 402, University of Wisconsin-Madison, October 1980.

Boral, H., and DeWitt, D.J., "Processor Allocation Strategies for Multiprocessor Database Machines," ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, pp 227-265.

Britton Lee, Inc. "IDM 500 Intelligent Database Machine," Product Announcement, 1980.

Canaday, R.E., Harrison, R.D., Ivie, E.L., Ryder, J.L., and Wehr, L.A., "A Backend Computer for Database Management," Communications of the ACM, 17, 10, October 1974.

Cesarine, D., De Luca, C., and Soda, G., "An Assessment of the Query-Processing Capability of DBMAC," In Advanced Database Machine Architecture, David K. Hsiao (ed.), Prentice Hall, Englewood Cliffs, New Jersey, 1983.

Comer, Douglas, "The Ubiquitous B-Tree," ACM Computing Surveys, Vol. 11, No. 2, June 1979.

Copeland, C.P., Lipovski, G.J., and Su, S.Y.W., "The Architecture of CASSM: A Cellular System for Non-Numeric Processing," Proceedings of the First Annual Symposium on Computer Architecture, December 1973.

Cullinane, J., Goldman, R., Meurer, T., and Navarawa, R., "Commercial Data Management Processor Study," Cullinane Corp., Wellesley, Mass., December 1975.

Drawin, M., and Schweiß, H., "A Performance Study On Host - Backend Communication," In Database Machines, H.O. Leilich and M. Missikoff (eds.), Springer Verlag, Berlin, 1983.

Date, C.J., An Introduction to Database Systems Volume II, Addison Wesley, Reading, Mass., 1983.

DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Transactions on Computers, June 1979.

DeWitt, D.J., and Hawthorn, P.B., "A Performance Evaluation of Database Machine Architectures," Proceedings of the Seventh International Conference on Very Large Databases, September 1981, pp 199-213.

Gordon, G., The Application of GPSS V to Discrete Systems Simulation, Prentice Hall, Englewood Cliffs, New Jersey, 1975.

Hanson, J.G., and Orooji, A., "RRDS - A relational Replicated Database System," Technical Report, CS-TR-86-07, University of Central Florida, Orlando, Florida, February 1986.

Hanson, J.G., and Orooji, A., "A Data Manipulation Language For A Relational Replicated Database System (RRDS)," Unpublished Technical Report, University of Central Florida, Orlando, Florida, May 1986.

Hanson, J.G., and Orooji, A., "Query Processing In A Relational Replicated Database System (RRDS)," Unpublished Technical Report, University of Central Florida, Orlando, Florida, June 1986.

Hanson, J.G., and Orooji, A., "Experiments With Data Access And Data Placement Strategies For Multi-Computer Database Systems," Proceedings of The Fifth International Workshop On Database Machines, Karuizawa, Japan, October 1987.

Hanson, J.G., and Orooji, A., "A Taxonomy Of Database Systems," Technical Report CS-TR-87-10, University of Central Florida, Orlando, Florida, June 1987.

Hanson, J.G., and Orooji, A., "Experiments With Hardware Organizations And Software Structures For Multi-Computer Database Systems," submitted to Information Sciences - An International Journal (Special Issue On Database Systems), 1988.

Haran, B., and DeWitt, D.J., "Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines," Pages 166-187, Database Machines, Edited by Leilich, H.O., and Missikoff, M., Springer-Verlag, 1983.

Hawthorn, P.B., and DeWitt, D.J., "Performance Analysis of Alternative Database Machine Architectures," IEEE Transactions On Software Engineering, Vol. SE-8, No. 1, January 1982, pp 61-75.

Horowitz, E., and Sahni, S., Fundamentals of Data Structures, Computer Science Press, Rockville, Maryland, 1982.

Hsiao, D.K., and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC-TR-81-7, The Ohio State University, Columbus, Ohio, July 1981.

Hsiao, D.K., and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part II)," Technical Report, OSU-CISRC-TR-81-8, The Ohio State University, Columbus, Ohio, August 1981.

Hsiao, D.K. (ed.), Advanced Database Machine Architecture, Englewood Cliffs, New Jersey, 1983.

Kerr, Douglas, S., "Database Machines With Large Content-Addressable Blocks and Structural Information Processors," IEEE Computer, Vol. 12, No. 3, March 1979.

Korth, H.F., and Silberschatz, A., Database System Concepts, McGraw Hill Book Co., New York, New York, 1986.

Leilich, H.O., Stiege, G., and Zeidler, H.Ch., "A Search Processor for Database Management Systems," Proceedings Fourth Conference On Very Large Databases, 1978.

Leilich, H.O., and Missikoff, M., Database Machines, Springer Verlag, Berlin, 1983.

Lin, S.C., Smith, D.C.P., and Smith, J.M., "The Design of a Rotating Associative Memory for Relational Database Applications," ACM Transactions on Database Systems, 1, 1, March 1976.

Maryanski, F.J., Fisher, P.S., Housh, R.D., and Schmidt, D.A., "A Prototype Distributed DBMS," Hawaii International Conference on System Sciences, January 1979, Vol. 2.

Maryanski, F.J., "Backend Database Systems," Computing Surveys, 12, 1, March 1980.

Missikoff, M., and Terranova, M., "The Architecture of a Relational Database Computer Known as DBMAC," Chapter 4, Advanced Database Machine Architecture, Edited by Hsiao, D.K., Prentice-Hall, 1983.

Mitchell, R.W., "Content Addressable File Store," Proc. Online Database Technology Conference, Online Conferences Ltd., England, April 1976.

Ozkaranhan, E.A., Schuster, S.A., and Smith, K.C., "RAP - Associative Processor for Database Management," AFIPS Conference Proceedings, Vol. 44, 1975.

Ozkaranhan, E.A., Schuster, S.A., and Sevcik, K.C., "Performance Evaluation of a Relational Associative Processor," ACM Transactions on Database Systems, June 1977.

Ozsu, T., "The Modeling and Analysis Of Distributed Database Concurrency Control Mechanisms", Ph.D. dissertation, The Ohio State University, Columbus, Ohio, 1983.

Pritsker, A.A.B., Modeling And Analysis Using Q-Gert Networks, Halsted Press and Pritsker And Associates, Inc., New York, 1977.

Pritsker, A. Alan B., Introduction To Simulation and SLAM II, John Wiley and Sons, New York, 1986.

Qadah, G., "Database Machines - A Survey," AFIPS Conference Proceedings, Vol. 54, 1985.

Qadah, G., and Irani, K.B., "Evaluation of Performance of The Equi-Join Operation On The Michigan Relational Database Machine," Proceedings IEEE Conference On Parallel Processing, 1984.

Qadah, G., and Irani, K.B., "A Database Machine For Very Large Relational Databases," IEEE Transactions On Computers, Vol. C-34, No. 11, November 1985.

Rothenie, J., Bernstein, P., Fox, S., Goodman, N., Hammer, M., Laders, T., Reeve, C., Shipman, D., Wong, E., "Introduction to a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems, 5, 1, March 1980.

Salza, S., Terranova, M., and Velardi, P., "Performance Modeling of the DBMAC Architecture," In Database Machines, H.O. Leilich and M. Missikoff (eds.), Springer Verlag, Berlin, 1983.

Samari, N.K., and Schneider, M.G., "A Queuing Theory Based Analytical Model of a Distributed Computer Network," IEEE Transactions on Computers, Vol. C-29, No. 11, November 1980.

Saver, C.H., and Chandy, K.M., Computer Systems Performance Modeling, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

Schiffner, G., Scheuermann, P., Seehusen, S., and Weber, H., "On A Specification And Performance Evaluation Model For Multicomputer Database Machines," In Database Machines, H.O. Leilich and M. Missikoff (eds.), Springer Verlag, Berlin, 1983.

Schuster, S.A., Nguyen, H.B., Ozkarahan, E.A., and Smith, K.C., "RAP.2 - An Associative Processor for Databases and its Applications," IEEE Transactions on Computers, C-28, 6, June 1979.

Shibayama, S., Kakuto, T., Miyazai, N., Yakota, H., Murakami, K., "A Relational Database Machine With Large Semiconductor Disk and Hardware Relational Algebra Processor," New Generation Computing, Vol. 2, 1984.

Smith, D., and Smith, J., "Relational Database Machines," IEEE Computer, Vol. 12, No. 3, March 1979.

Srinidhi, H.N., "A Relational Database Machine Using Magnetic Bubble Memories," Ph.D. Dissertation, Southern Methodist University, 1982.

Stonebraker, M., Wong E., Kreps, P., and Held, G., "The Design And Implementation of INGRES," ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976.

Stonebraker, M., and Neuhold, E., "A Distributed Database Version of INGRES," Proceedings of the Second Berkeley Workshop on Distributed Databases and Computer Networks, May 1976.

Stonebraker, M., "A Distributed Database Machine," Memorandum No. UCB/ERL M78/23, Electronics Research Laboratory, University of California, Berkeley, May 1978.

Stonebraker, M., "MUFFIN: A Distributed Data Base Machine," Memorandum No. UCB/ERL M79/28, Electronic Research Laboratory, University of California, Berkeley, May 1979.

Su, S.Y.W., and Lipovski, G.J., "CASSM: A Cellular System for Very Large Databases," Proceedings of the VLDB Conference, 1975.

Teradata Corporation, "Teradata DBC 1012," Product Announcement, Los Angeles, 1986.

Ullman, J.D., Principles of Database Systems, Computer Science Press, Rockville, Maryland, 1982.

Weiderhold, G., Database Design, McGraw Hill Book Co., New York, New York, 1977.

Weinburg, Victor, Structured Analysis, Yourdon Press, New York, 1979.

Williams, R., "R*: An Overview of the Architecture," IBM Research Report RJ3325, December 1981.